Implementation of Constrained Control Allocation Techniques Using an Aerodynamic Model of an F-15 Aircraft

by

John Glenn Bolling

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Aerospace Engineering

Approved:

Dr. Wayne Durham, chair

Dr. Mark Anderson

Dr. Frederick Lutze

May, 1997

Blacksburg, Virginia

Key Words: Flight Control, Control Allocation, F-15

IMPLEMENTATION OF CONSTRAINED CONTROL ALLOCATION TECHNIQUES USING AN AERODYNAMIC MODEL OF AN F-15 AIRCRAFT

by

John Glenn Bolling

Chairman: Dr. Wayne Durham

Aerospace and Ocean Engineering Department

(ABSTRACT)

Control Allocation as it pertains to aerospace vehicles, describes the way in which control surfaces on the outside of an aircraft are deflected when the pilot moves the control stick inside the cockpit. Previously, control allocation was performed by a series of cables and push rods, which connected the 3 classical control surfaces (ailerons, elevators, and rudder), to the 3 cockpit controls (longitudinal stick, lateral stick, and rudder pedals). In modern tactical aircraft however, it is not uncommon to find as many as 10 or more control surfaces which, instead of being moved by mechanical linkages, are connected together by complex electrical and/or hydraulic circuits. Because of the large number of effectors, there can no longer be a one-to-one correspondence between surface deflections on the outside of the cockpit to pilot controls on the inside. In addition, these exterior control surfaces have limits which restrict the distance that they can move as well as the speed at at which they can move. The purpose of Constrained Control Allocation is to deflect the numerous control surfaces in response to pilot commands in the most efficient combinations, while keeping in mind that they can only move so far and so fast. The implementation issues of Constrained Control Allocation techniques are discussed, and an aerodynamic model of a highly modified F-15 aircraft is used to demonstrate the various aspects of Constrained Control Allocation.

This work was conducted under NASA research grant NAG-1-1449 supervised by John Foster of the NASA Langley Research Center

Acknowledgements

I would like to express my appreciation to all those who helped to contribute to this research. For serving as my advisory committee, and providing their expertise and other words of wisdom, I thank Dr. Wayne Durham, Dr. Mark Anderson, and Dr. Fred Lutze. I thank Nate Fower and Jim Buckley of McDonnell Douglas Aerospace for offering various debugging hints and other necessary data during my development of the F-15 ACTIVE Control Allocation Database. I am also grateful to my girlfriend, Heather, for taking time out of her schedule to proofread this thesis and offer some constructive criticism.

John G. Bolling

CONTENTS

FIGURES AND TABLES

CHAPTER 1.

Introduction

The primary idea behind the design of aircraft flight control surfaces has been to position them in such a way that they function primarily as moment generators, allowing 3 types of rotational motion (roll, pitch, and yaw). Classically, these 3 degrees of freedom were manipulated by 3 primary control surfaces. The ailerons, located on the trailing edges of the wings, were designed to operate differentially, thus causing the aircraft to roll. The elevators, located on the trailing edge of the horizontal tail, were used to change the pitch attitude of the aircraft. Finally, the rudder, located on the vertical tail, could be deflected in such a way to cause the aircraft to yaw. In these classical designs, the control surfaces were generally directly connected to pilot controls inside the cockpit.

It is also easy to see that if the control effectiveness is known for each of the 3 control surfaces mentioned above, then the classical 3 control/3 degree of freedom system can be defined by an algebraic problem with 3 equations (the commanded moments) and 3 unknowns (the control deflections). Assuming that the mathematical system of equations is consistent, then a unique control configuration exists for any desired vector of control-generated moments.

While modern aircraft may have the 3 conventional sets of aircraft control surfaces, they are generally designed to operate independently of one another. The ailerons for instance, may be able to deflect both symmetrically and differentially, allowing some pitch control in addition to the conventional roll and yaw effects. The elevators on modern tactical aircraft are now divided into left and right stabilators, which in addition to deflecting symmetrically, can operate in a differential sense as well. Other nonconventional flight control surfaces such as canards and thrust vectoring have also been introduced. In addition, with the development of the modern fly-by-wire control system architectures, the mechanical linkages between control surfaces and cockpit inceptors have been made obsolete. Thus, the problem of finding the required combinations of control deflections for a given

vector of commanded aircraft moments is no longer the previously mentioned algebraic system with one solution, but is an under-determined system having an infinite number of solutions, provided that the 3 moments can be attained.

With so many possible combinations of control deflections for any attainable set of moments, the question of how to allocate the controls in response to pilot commands no longer has an obvious answer. This question has recently been the topic of a great deal of research and development and has led to many techniques including the constrained control allocation algorithms presented here. In answering this question, Constrained Control Allocation has been developed to utilize the controls' maximum capabilities in generating moments such that a solution to any attainable moment vector will result in the most efficient use of control deflections. While the underlying theory behind these techniques is described in detail in References 1 and 2, it will be paraphrased here.

In order to understand the nature of Constrained Control Allocation, it is imperative to think of the control surfaces on an aircraft as generating some *m* dimensional control space where *m* represents the number of aircraft controls. Each control surface generally has a maximum and minimum deflection limit which is determined either by the control's physical limitations or by some aerodynamic constraint. Within the control space then, there exists an admissible subset of controls (Ω), which represents all possible combinations of controls that do not violate any constraints.

In addition to the control surface limits, each control also has some effectiveness in generating the 3 aircraft moments. Therefore, a 3 by *m* control effectiveness matrix (**B**) can be defined which serves as a linear mapping between the *m*-Dimensional admissible subset of controls and the 3-Dimensional subset of attainable moments (Φ). The problem of allocating controls then consists of extending the commanded moment vector to the boundary of Φ, and finding the point of intersection. Using concepts of linear mappings, the (unique) vector of controls which map to this point on the boundary can be found, and can then be scaled to achieve the desired moment magnitudes. Of course, if the commanded moment is not attainable, (ie. the moment is somewhere outside of Φ), then the resulting control configuration is not scaled, and a solution is returned which at least, produces a moment vector that points in the commanded direction. Note that one of the unique aspects of this control

allocation scheme is that none of the defined control constraints will ever be violated.

1.1 Background

Research into the Constrained Control Allocation methods was originally stimulated by a broader field of research. This research involved the design and structure of modern aircraft control laws that would allow the pilot to fly with “reckless abandon” in high angle of attack flight regimes, having no concern for the possibilities of aircraft departure.

This design philosophy is based on a generalized structure allowing a modular architecture and is shown in Figure 1.1. In this figure, the control stick logic receives the required information from the control law to simulate a control stick “feel” that provides satisfactory flying qualities. In addition, control surface rate information is fed back from the control allocation scheme so that the stick deflection rates can be controlled, thus preventing the pilot from exceeding the aircraft’s capabilities for any particular flight condition. The logic then uses the pilot inputs to calculate some desired aircraft behavior such as desired roll rate, pitch rate, or normal acceleration to be sent down stream to the control law.

The proposed control law type is one of the model-following or dynamic inversion schemes since they are potentially capable of operating in a wider range of flight conditions than the classical feedback designs. The control law also receives information about the current controls’ moment generating capabilities so that control stick deflection can be synchronized to the aircraft’s current capabilities. That is, full stick would imply that the aircraft should perform its desired behavior (like pitch acceleration) to its maximum capabilities while half stick would result in half of those desired capabilities. The control law then takes the desired aircraft behavior and converts it to a set of required control-generated moments using the aircraft equations of motion and an onboard aerodynamic database or aircraft simulation.

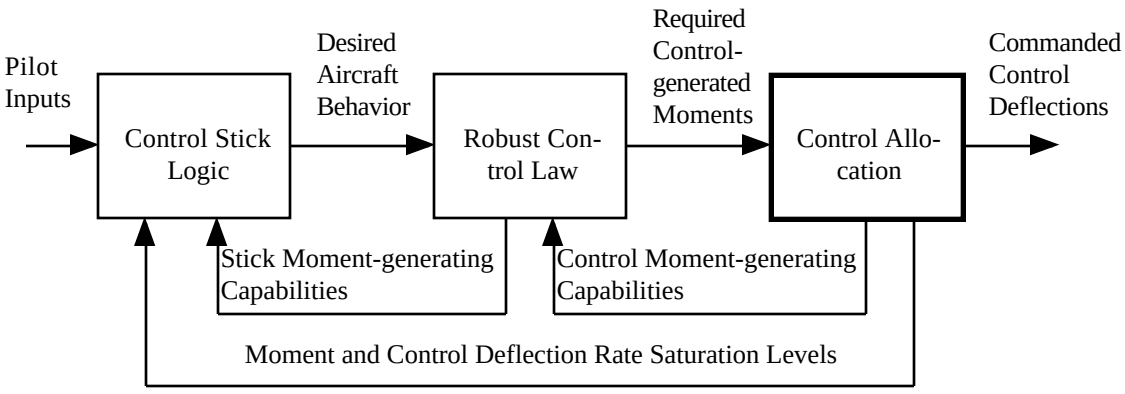

Figure 1.1 A "Reckless Abandon" Control Stick/Law/Allocation System
This diagram represents a preliminary design architecture for a modular aircraft control system. The blocks representing the control stick logic, control law, and control allocation scheme are generalized to allow easy implementation of new and improved ideas.

The control allocation scheme then uses the desired moments to find a set of admissible control deflection commands to achieve those moments. These commands should not violate any prescribed position or rate limits. The subject of this thesis is based on the third block of Figure 1.1 (Control Allocation), and how it should be implemented. It is not meant to be a complete description of the Constrained Control Allocation theory. A theoretical background can be referenced from a few of the numerous papers and journal publications published during this research. (The author's suggested reading list includes references 1 and 2 for a theoretical introduction followed by references 3, 4, and 5.) Some of the theory, as it applies to various implementation issues, is presented.

1.2 Research Objectives

This Thesis covers a rather diverse area of research which includes many objectives. The primary goals were to develop a generic constrained control allocation architecture to allow quick and easy implementation of different aircraft configurations. At the same time, the numerical robustness and functional abilities of these algorithms were also improved.

In previous work at Virginia Tech, these algorithms were tested using control effectiveness data and control constraints representative of modern aircraft. Reference 6 describes one of the early simulation implementations involving actual aircraft data, and reference 5 builds upon this implementation with the introduction of non-linear effectiveness and moment rate allocation. The results from these works prompted a renewed interest in constrained control allocation from industry and led to the secondary objectives of this research, which include implementing the algorithms into the F-15 ACTIVE (Advanced Control Technologies for Integrated VEhicles) flight control laws for further testing.

The F-15 ACTIVE aircraft is a research project co-developed and supported by McDonnell Douglas Aerospace, NASA Dryden Flight Research Laboratory, Pratt and Whitney, and the United States Air Force, in an attempt to demonstrate new control law philosophies and thrust vectoring technologies. Virginia Tech signed a cooperative agreement with McDonnell Douglas Aerospace that included the study of control allocation as a

part of the F-15 ACTIVE program. This research can be subdivided into 3 phases. Phase 1 consists of extracting and formatting the non-linear control effectiveness data for the 9 aerodynamic and thrust vectoring controls to fit within the control allocation architecture, and testing the algorithms using the acquired data in batch simulations. Phase 2 of this research includes plans to implement the developed code and data into a six degree of freedom, piloted simulation to analyze real-time and flying quality aspects. Ultimately, phase 3 will consist of programming the algorithms into the F-15 ACTIVE flight control computers for actual flight testing. The research documented here marks the completion of the phase 1 effort.

1.3 Comments and Suggestions to the Reader

The primary focus of this thesis is the implementation issues of control allocation. As a result, there are no theories or ideas presented here that have not been mentioned in the references cited previously. Any theory presented is only discussed in terms of implementation and software development issues. Because of the broad area of research covered, the author feels that the best presentation format is to discuss the work in chronological order. Unfortunately, this format results in a document which contains guidelines and theory dispersed throughout. It is therefore suggested that the reader who is only interested in implementing control allocation into different aircraft models, should read Chapters 2, 7, 8, and Appendices I-III. These sections contain most of the guidelines that should be followed, as well as the code listings written in FORTRAN. Chapters 3 - 6 discuss some of the theory behind the features of control allocation and should be read by those who desire a more in depth understanding of the current control allocation algorithms.

The author understands that there exist an infinite number of possible control allocation schemes for any aircraft having more controls than degrees of freedom. However, when the terms “control allocation” are mentioned here, it should be understood that the author is referring to the Constrained Control Allocation (sometimes referred to as Direct Control Allocation) methods developed at Virginia Tech. Any exceptions to these conventions will otherwise be explicitly stated.

CHAPTER 2.

Data Collection

Like any control allocation algorithm, the constrained control allocation techniques discussed here require knowledge of the controls' effects on aircraft moment coefficients. The three coefficients of interest are C_l (roll), C_m (pitch), and C_n (yaw). In addition, data may also be included which can be used to minimize a particular objective as a function of control surface deflections. Other data requirements specific to constrained control allocation techniques include control minimum and maximum position limits and actuator rate limits. These data may also be dependent on other variables. As an example, many control laws may have software constraints, determined by the dynamic pressure, which are imposed on the available surface deflections. It is also obvious that the control actuators are not able to maintain a constant rate across the entire flight envelope, since hinge moments can change quite radically. Although in the current implementations, rate limits are assumed constant, there have been some experiments involved with adjustable position limits, particularly, those associated with thrust vectoring on the F-15 ACTIVE aircraft. The purpose of this chapter is to discuss some of the preferred data format methods with some of the lessons learned, and conclude with a description of the data collection utilities developed for this implementation.

2.1 Data Format and Dependencies

Since the fundamental theories behind constrained control allocation methods involve such concepts as linearity and linear mappings[1], some care must be taken to ensure that the control effectiveness data fits a suitable format. One of the most important assumptions is that any two controls' deflection capabilities are uncoupled from one another. The left and right stabilators on the F-15 ACTIVE for example, are completely isolated such that one can move without regard to the other, allowing both symmetric and/or differential horizontal tail deflections. By applying the position constraints to this two degree of freedom

system, a subset of admissible control deflections (Ω) can be found. A typical figure of what this subset may look like in control space for two controls is shown in Figure 2.1. Two important insights can be gathered from this figure. First of all, not only are the controls uncoupled from one another inside the constraints, but they are also uncoupled at the constraints. This property allows one control to be held at one of its constraining values while the other is free to move between its maximum and minimum constraints, resulting in a move along the edge of the constraining "box". Second, the linear independent nature of the controls implies that any point in Ω can be expressed as a linear combination of the two controls, just as a point in any vector space can be represented by a linear combination of its basis vectors. While the consequences of violating these facts will be discussed later, it should be pointed out that as long as these conditions are met, more controls can be added, resulting in higher dimensional admissible subsets. Using two controls, Ω can be described by a 2-Dimensional "box". When a third independent control is added to the system, Figure 2.1 would be best described as a "cube". Although harder to visualize, larger numbers of controls generate higher dimensional "hypercubes", yet the same linear concepts apply as with the simple 2-Dimensional case.

As mentioned earlier, the fact that any point in Ω can be uniquely expressed as a combination of the individual control surfaces, gives rise to the most preferred (and most convenient) method of gathering data. This format involves specifying the dependent data in terms of the individual left and right control deflections subject to their respective minimum and maximum constraints. Unfortunately, many existing databases think of controls as being symmetric and differential, with definitions given below in equations 2.1 and 2.2.

Symmetric Portion: $$\sigma = (\delta_l + \delta_r)/2 \tag{2.1}$$

Differential Portion: $$\delta = \delta_l - \delta_r \tag{2.2}$$

One might first think to use the data in this form since it is most readily available in the aerodynamic databases. However, when implementing non-linear data, this method carries the burden of having to add an unnecessary table lookup dimension to the interpolation routines.

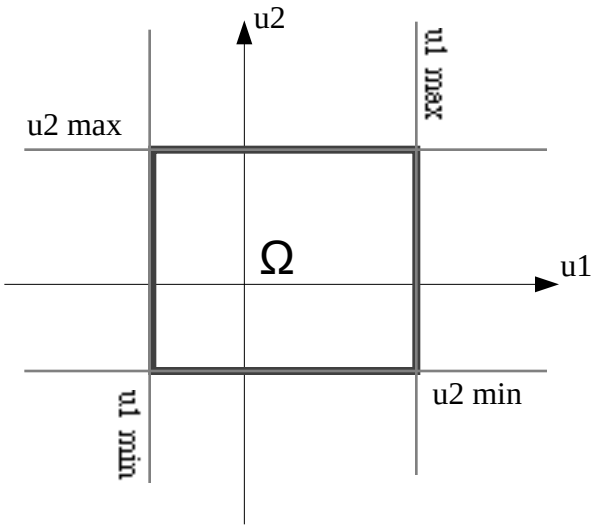

Figure 2.1 A Typical Subset of Admissible Controls

For instance, extracting the pitching moment effectiveness due to symmetric stabilator only, would require an interpolation using both the current symmetric and differential deflections. This extra step is required because of the coupling between the symmetric/differential deflection conventions. That is, specifying either a symmetric or differential deflection alone does not map to a unique left/right control configuration. In contrast, gathering effectiveness data for a left or right control surface only requires that the current position of the control in question be known (assuming that there are no interaction effects from the opposing control).

Perhaps the best incentive of avoiding this type of data format is the fact that the subset of admissible controls given in terms of symmetric deflection (σ), and differential deflection (δ), is defined by Figure 2.2. Note that because of the "diamond shape" nature of this figure, the constraint lines cannot be followed without having to vary both symmetric and differential controls. Thus, the controls are coupled along the constraints. There are also other instances when inconveniences such as this occur, and special steps need to be taken to avoid them. (Discussion of these will be saved until Chapter 8 when the F-15 thrust vectoring limits are presented). Furthermore, when allocating symmetric and differential control deflections, the process of adapting to individual control failures becomes somewhat problematic.

In early control allocation implementations, the nature of control effectiveness was assumed linear and was generally interpolated as functions of Mach number and angle of attack only. This assumption was known to be a rather simplistic approximation, yet, when small deflections were commanded, it seemed to produce satisfactory results. Upon extending the direct allocation methods to include actuator rate capabilities[4], a more accurate database could be utilized. These algorithms allocate controls during each sample frame, using as constraints, the most restrictive of either the amount of deflection that an actuator can produce in that given frame (determined from its rate capabilities), or the control's global position limits. The allocated changes in control deflections then produce the desired change in moment coefficients. Since these algorithms reset the origin in control space for each sample frame, it makes sense to include control effectiveness as a function of control deflection as well as other aircraft states. These properties are shown in Figure 2.3.

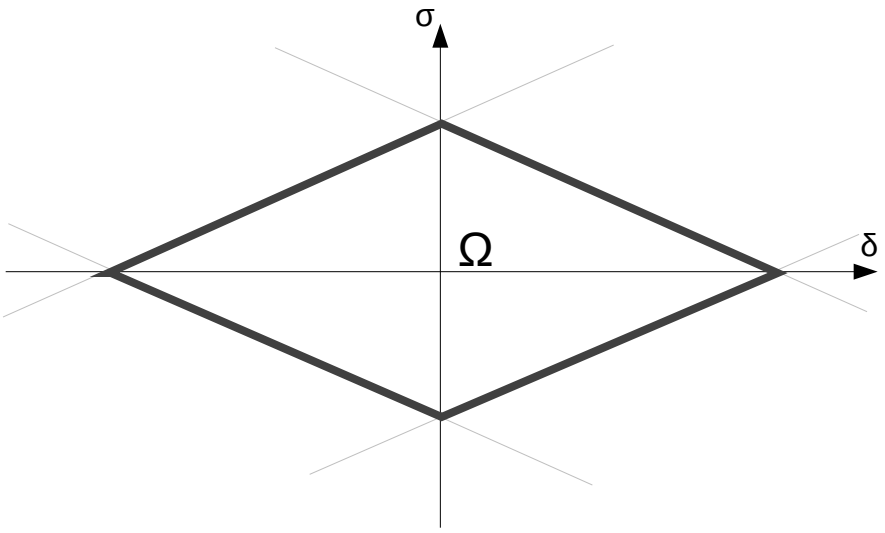

Figure 2.2 A Typical Subset of Admissible Controls Using Symmetrical vs. Differential Coordinates

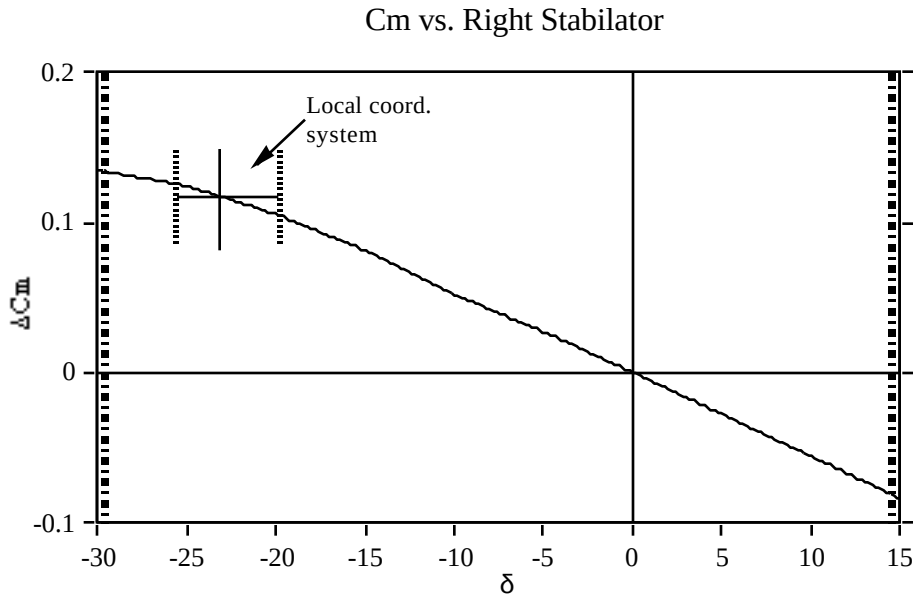

Figure 2.3 A Typical Plot of Moment Coefficient vs Right Stabilator Deflection
Shown with this plot is a local coordinate system resulting from the frame wise limits imposed on Control Allocation with Rate Limiting. The constraints on this system represent the actuator rate capabilities. The constraints on the larger coordinate system represent the global position limits. Note that as the global position changes, the local coordinate system moves along this curve. Effectiveness data is taken as the slope of the Cm vs. δ curve at the origin of the local coordinate system. This data is based on the F-15 ACTIVE aerodynamic database for a Mach number of 0.4, angle of attack of 8.0 deg. and an altitude of 20000 ft.

Because of the abilities to handle non-linearities with respect to control deflections, the rate limiting algorithms allow the inclusion of other dependent data such as drag effects, which were to non-linear (with respect to control deflection), to give valid results using the early control allocation methods. These other effects can be used as additional objectives that the allocation algorithms can optimize.The optimization aspects will be demonstrated later where control allocation with rate limiting is used to minimize control-induced drag while obtaining the commanded moments. There is no limit to the type of minimizing objectives that can be included. However, there are a few restrictions to the use of such objective data. First, only one objective can be utilized at any given time. While a control allocation database may contain data for objectives such as minimizing drag and minimizing hinge moments, only one (or some weighted combination of the two objective functions) can be optimized. Thus, the control law may need some type of switching function to instruct the control allocation algorithms when to switch from one objective to another. Second, the objectives should be in a form such that they can be expressed as continuous functions of control deflections.

Current control allocation algorithms also require linear control effectiveness data for the purpose of calculating attained moments. Two methods that have been attempted involve the global slope method, which bases the control effectiveness on non-linear data at zero deflections, and the secant-slope method, which calculates the slope of the line drawn between the minimum and maximum control induced moment. These two methods are demonstrated graphically in Figure 2.4. Notice that the secant slope method produces accurate results at the control constraints but generates errors at other control positions. Likewise, the global slope method gives fairly accurate results for small deflections at the price of obtaining less accurate results for large deflections. Of course, the terms “large” and “small” are relative terms and are influenced by how non-linear the data is. Other possibilities that have not been investigated include using the existing aerodynamic table look-ups to find the moment contributions due to the controls. This method will provide results as accurate as the results from which the tables were constructed, but will consume a great deal of computing cycles. In the control allocation algorithms implemented, the global slope method is the preferred choice since the non linear effectiveness tables already contain the data needed, thus alleviating some of the storage requirements.

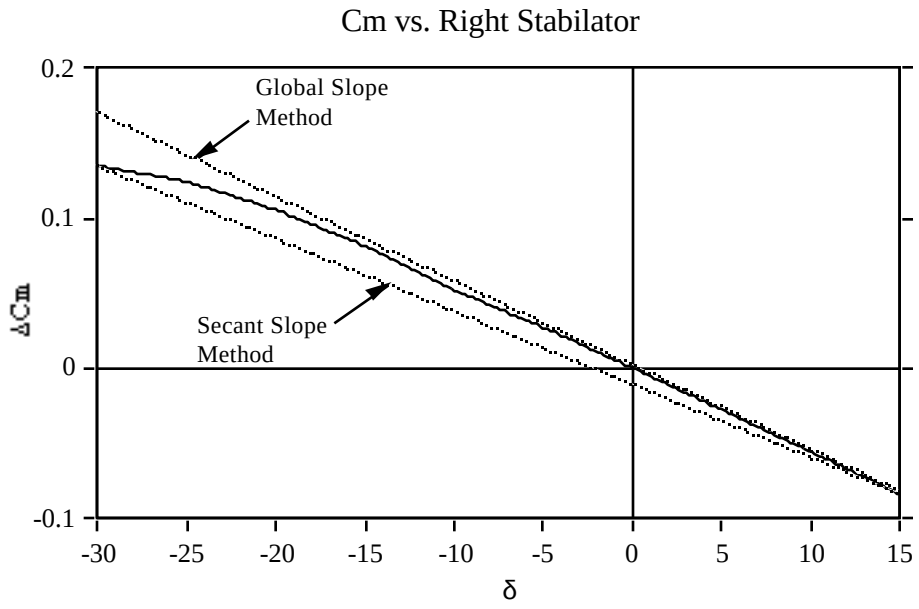

Figure 2.4 Different Methods to Find Linear Control Effectiveness Data

2.2 Data Interpolation Methods

Due to the large computational requirements of these control allocation algorithms, the speed of the methods used to interpolate control effectiveness data are important. In this section, different types of interpolation methods available are discussed in terms of speed, storage requirements, and other problems that have been discovered. At present, two different techniques have been investigated for various control allocation implementations. In the early experiments, where linear control effectiveness was assumed[6], the data were acquired using third and fourth order polynomial curve fits across the known aerodynamic table break-points to account for the changes in control effectiveness due to changes in aircraft states like angle of attack and Mach number. In addition, all of the polynomial coefficients were stored off-line and loaded into computer memory during an initialization pass. Another method which is currently implemented is very similar to the curve fit methods except that the interpolation functions are treated as being affine. That is, any 2-D planar slice of the data results in a linear function. With this type of data extraction scheme, several affine functions must be generated for a range of independent parameters to account for any non-linearities. Just like the curve fit methods, the coefficients for these functions are also stored off-line and then loaded into memory during an initialization pass. The standard linear interpolation methods employed in most simulation environments has been avoided here because of their large computational demands for multi-dimensional data sets.

The curve fit methods proved to be very desirable in terms of memory requirements when there were relatively few data dependencies. Unfortunately, the method of generating these polynomial curve fits begins to break down as the number of independent parameters increases. As an example, consider the drag increments due to the left stabilator for the F-15 ACTIVE shown in Figure 2.5. After inspection of the original data, it was decided that a quadratic should provide sufficient accuracy. The resulting function is plotted along with the actual values returned from the aerodynamic database, showing a smooth and well matched curve. This information can be carried along in the simulation if desired (it only requires storing 3 real number coefficients).

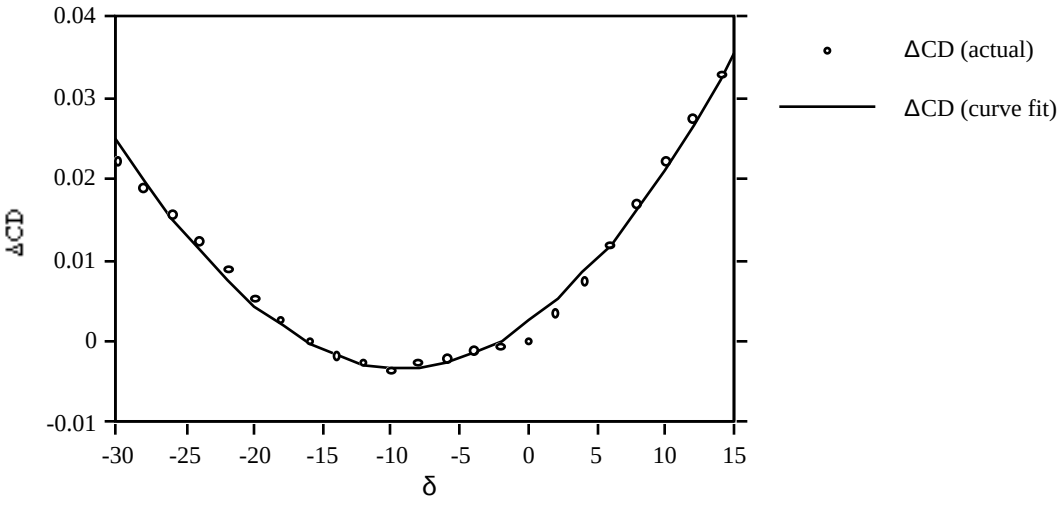

Figure 2.5 Increments in Drag Coefficient Due to Left Stabilator Deflection
Flight conditions for this data are for a Mach number of 0.8, an Angle of Attack of 8.0 deg. and an altitude of 20000 ft. The curve fit results are for a quadratic function {Cd = (6.5 x 10^-5)δ^2 + (1.2 x 10^-3)δ + 2.4 x 10^-3} where control deflection is measured in degrees.

On the other hand, if it is only the control effectiveness that is needed, then this function can be differentiated with respect to control deflection off-line and the amount of required storage then reduces to only two real coefficients.

Of course, this simple case only accounts for the non-linearities in control position at one particular flight condition. In general, there may be as many 4 dependencies on drag, which result in a more complicated function. The drag effectiveness lookup for the F-15 ACTIVE thrust vectoring nozzles for instance, require knowledge of nozzle pressure ratio, Mach number, angle of attack, and nozzle deflection. Recall that in Figure 2.5, the general shape of the data was visually inspected first, and an appropriate function was chosen. Obviously, this method of picking functions fails when the dependencies are higher dimensional, since the ability to view the shapes of such functions becomes nearly impossible. An alternative is to generate curve-fit polynomials for each 2-Dimensional "slice" of data and perform a linear interpolation between them when the independent parameters do not lie on one of these 2-Dimensional planes. This procedure, however, reintroduces the problem of slow data interpolation due to computationally intensive table lookup algorithms.

What is required then is a way of utilizing the speed benefits obtained when saving data as an analytical function while avoiding the necessity of having to know the general nature of the function prior to performing the curve fits. This form of interpolation is best handled using affine functions of the independent variables. For example, a function F(x,y) can be represented in this fashion by:

$$F(x,y) = C_1xy + C_2x + C_3y + C_4 \tag{2.3}$$

This function then has the special form that any planar slice (ie. setting all but one of the independent parameters to a constant), reduces to a linear function of the remaining parameter. As a result, this interpolation method gives the same results as the higher dimensional linear interpolation methods commonly seen in aerodynamic table look-up routines, yet reduces the time required to do the interpolation since there is a significant reduction in the required floating point operations[7]. The drawback to this method however is that the storage requirements, (the mesh constants C_1, C_2, C_3, and C_4 for each table "block" must be stored), become quite large as the dimension of the table increases. It can be shown that the

number of mesh constants per table block increases according to 2^n where n is the number of independent parameters. Reference 7 performs some interesting tests to evaluate the tradeoffs associated with this method and the standard linear interpolation methods. The conclusions gathered from this paper suggest that the affine interpolation methods be avoided when the number of independent variables exceeds three.

In the current implementations, the affine interpolation method is used to gather control effectiveness data. All of the data for the F-15 ACTIVE depends on 3 parameters, while the two thrust vectoring tables contain an additional independent parameter describing the effects due to nozzle pressure ratio. These 4-D cases are handled in a slightly different way, and will be discussed later. The next section will describe this interpolation technique in more detail.

2.3 The Affine Data Interpolation Procedure

Equation 2.3 gives the form of the affine function for a 2-Dimensional case. It is also assumed that the dependent data is known at incremental points throughout the 2-D table. (For the F-15 ACTIVE implementation, a utility was written in FORTRAN to interface with the aerodynamic table look-up routines, and perform the necessary independent variable sweeps). These known values will be referred to as nodes. The nodes for a representative table are depicted in Figure 2.6. From this figure, it can be seen that the nodes produce distinct subdivisions within the table (referred to here as blocks). These blocks can be labeled according to their row number and column number. That is, block (1,4) would be represented by the subdivision created by taking the intersection of row 1 with column 4.

It is now desired to find the coefficients for equation 2.3 for each block in the table. Note that for the 2-D case, there are 4 unknown coefficients which determine the equation for any block. These blocks in turn, are defined by 4 known node points. Thus, by applying equation 2.3 at each node, a system of 4 linear equations with 4 unknowns is formulated, and the coefficients for the particular block can be uniquely determined. The coefficients can then be stored off-line and loaded during an initialization pass.

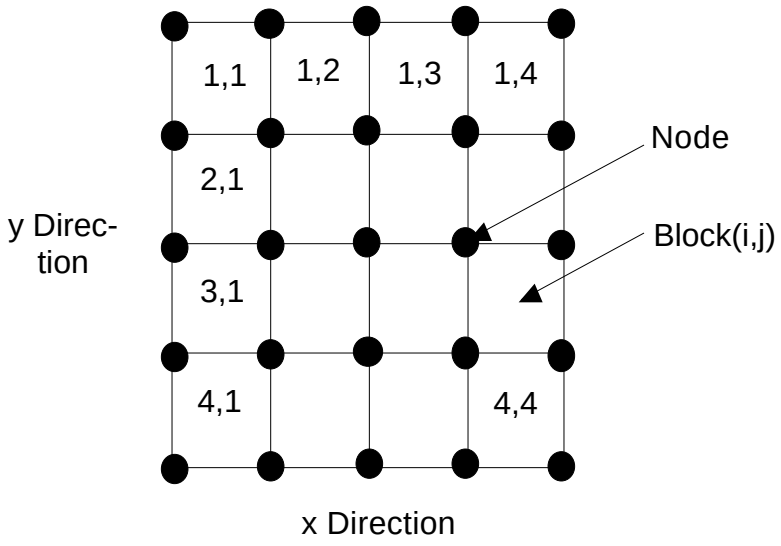

Figure 2.6 A 2-Dimensional Data Table Showing The Known "Nodes" and "Block" Subdivisions

The only remaining CPU operations required are finding out which block the current flight condition resides in, and using the corresponding mesh coefficients to calculate the dependent data from equation 2.3.

While this example applies to a 2-Dimensional table look-up, the techniques used can be applied to higher *n*-Dimensional tables as well since there will always be 2^n mesh coefficients and 2^n nodes per block. The current Matlab® utilities, developed for this research, can handle tables having as many as 3 dimensions. For the case of the thrust vectoring nozzles which require a 4-D table look-up, the 3-D data is generated for incremental values of the 4th parameter, and the simulation is required to perform one linear interpolation in the 4th parameter's direction.

In implementations prior to the F-15 ACTIVE, it was thought that the amount of storage required was actually half of what the table dimensionality indicated. This hypothesis was formulated because of the fact that only the derivatives with respect to control positions were desired, so that after differentiating functions of the form in equation 2.3, only the mesh constants multiplying terms containing the controls' deflections remained. Affine interpolation methods, however, only guarantee continuous functions across the entire table. The theory does not account for continuity in first (or higher) order derivatives as the independent variables move from one table block to another. As a result, the reduced data sets often led to discontinuous effectiveness data with respect to control deflections, and caused control chattering problems to occur in situations where the control positions were on the boundary between two table blocks. The solution to the problem was to calculate the derivative information before generating the mesh constants. This step is currently done in the FORTRAN "Sweep Data" utility using a 4th order central difference approximation to the derivative.

The utilities mentioned here have been written to provide a set of tools for extracting effectiveness data from a given aerodynamic look-up module, using the extracted data to generate the affine data interpolation constants, and saving the constants in data files so that the control allocation algorithms can use them. Furthermore, the algorithms used to interpolate the data have been modularized and isolated from the rest of the control allocation software. While the structure of the data interpolation functions that the control allocation

algorithms require will be described later, the intended goal here is to provide a standard, quick, and relatively easy method of developing appropriate data that can be plugged into the control allocation software with minimal effort.

2.4 The Data Extraction Utilities

This section describes the utilities that can be used to extract and format the control effectiveness data so that it can be processed by the current control allocation algorithms. The set of tools assumes that the affine data interpolation technique discussed earlier will be used. It is composed of a FORTRAN executable "Sweep Data", and three Matlab® scripts "MMCS2D", "MMCS3D", and "MAT2ASCII". The complete source code and documentation for these utilities can be found in Appendix I.

The "Sweep Data" utility is a FORTRAN executable program that was originally designed to interface the F-15 ACTIVE aerodynamic database and extract effectiveness information in a form suitable to that described earlier. However, the final project turned out to be a very useful and powerful tool, and so its features and source code are documented here with the goal of making the process of implementing future models faster and easier. The utility includes a command-line shell interface used to change various parameters, program flags, and to perform the data sweeps. The source documentation for these interface routines can be found in Appendix IV. In addition, the data obtained from the sweeps are exported as Matlab® workspace (*.mat) files, and require use of the Matlab® External Interface libraries[8].

The purpose of this utility is to allow the user to increment as many as three prescribed independent parameters between their minimum and maximum possible values and record the aerodynamic coefficients that are returned from the aerodynamic database. The powerful flexibility in the program is provided by a set of user-defined flags that alter the way data is recorded or calculated. Some of these options include:

1) The ability to enable or disable any flap scheduling functions: In some cases, the leading and trailing edge flaps may be treated as integral parts of the basic airframe. In such instances, it may be desired to let the flap scheduling functions set the flap

positions before the aerodynamic coefficients are calculated. At other times however, it may be desired to calculate the exclusive effects due to the controls without any flap interactions.

2) The ability to use right/left control deflections or symmetric/differential deflections: Many aircraft databases build up the aerodynamic coefficients using symmetric and differential control surface deflections based on the definitions in equations 2.1 and 2.2. This flag allows either convention to be used by using the equations to convert from one to the other.

3) The ability to record the total aerodynamic coefficients or increments due to a control only: When this utility is supported, Sweep Data can optionally record the intermediate control effects that are normally averaged with other terms. This feature allows a method of excluding all other control interaction effects that are normally built into the total aerodynamic coefficients returned by the database.

4) Derivative extracting: Normally, Sweep Data returns the increments in aerodynamic coefficients due to some change in an independent parameter. These data are useful for generating plots to see how effective a control is, but it does not provide the true effectiveness data required by control allocation. With this option enabled, the control effectiveness is extracted by using a fourth order central difference approximation to the derivative with respect to the independent parameter.

Other features include the ability to specify certain parameters as constant values other than zero for a particular data sweep, and the ability to adjust the increment size for each independent parameter. All of these features and their commands are displayed in a text menu within the shell interface so that the user does not have to memorize a set of complicated commands.

To demonstrate the capabilities of this utility, two data sweeps have been plotted. The first example shows an angle of attack and left canard sweep with Mach number and altitude held constant to 0.5 and 10000 feet. The stored dependent data is the increment in the pitching moment coefficient due to canard only (no interaction effects) and is plotted in Figure 2.7. Figure 2.8 shows data for the canard effectiveness in pitch (ie. $\partial Cm/\partial \delta$) generated by setting the derivative extraction flag. All independent parameters are exactly as

specified for the first sweep. It is this figure which represents the data required by the control allocation algorithms, and it must be converted into the mesh constant data for the affine interpolation tables. This conversion may be done using the Matlab® scripts MMCS2D or MMCS3D, which use the theory developed in section 2.3 to find the required constants by solving a system of linear equations for each table block. (See appendix I for specific details about these functions). Since Figure 2.8 represents a 2-Dimensional table, the MMCS2D script was used to calculate the mesh constants. Using these constants, the left canard was swept across its deflection range, holding the angle of attack at zero. The results, along with the original data for the zero angle of attack slice are plotted together in Figure 2.9, demonstrating the fact that the affine look-up procedure gives the same results as the standard linear interpolation techniques. As a final step, the mesh constant data could be converted to an ascii text file using the MAT2ASCII script so that the control allocation algorithms can read them. The details of this code are also contained in Appendix I.

This section, along with the source documentation in Appendix I, has outlined the basic procedure for creating a control effectiveness database for practically any aircraft model. The steps are summarized here:

1.) Interface the Sweep Data application with the desired aerodynamic database. This step will generally be the most time consuming process since the source code will require some slight modifications.

2.) Run the Sweep Data application, using the appropriate flags to generate the desired data.

3.) Generate the affine interpolation mesh constants for the data using the Matlab® scripts MMCS2D or MMCS3D.

4.) Export the mesh constants data to ascii files for inclusion in the control allocation algorithms using the Matlab® MAT2ASCII procedure.

The only remaining tasks are developing the aircraft specific control allocation modules to initialize the affine data, interpolate the data, and calculate control constraints. Discussion of these steps will be reserved for Chapter 7.

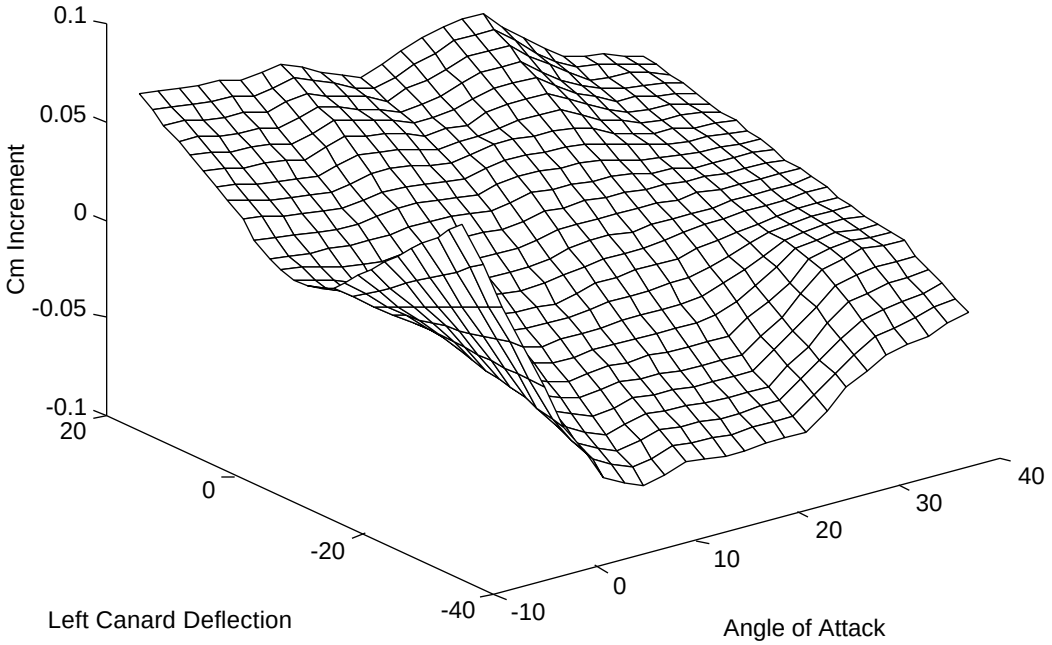

Figure 2.7 Cm Increments as a Function of Angle of Attack and Left Canard Deflection
This plot was generated using Matlab® and the FORTRAN "Sweep Data" utility linked with the F-15 ACTIVE Aerodynamic Database. Angle of Attack was swept between -10 and 40 degrees and Left Canard was swept from -35 to 15 degrees. Reference Mach number is 0.5 and reference Altitude is 10000 ft.

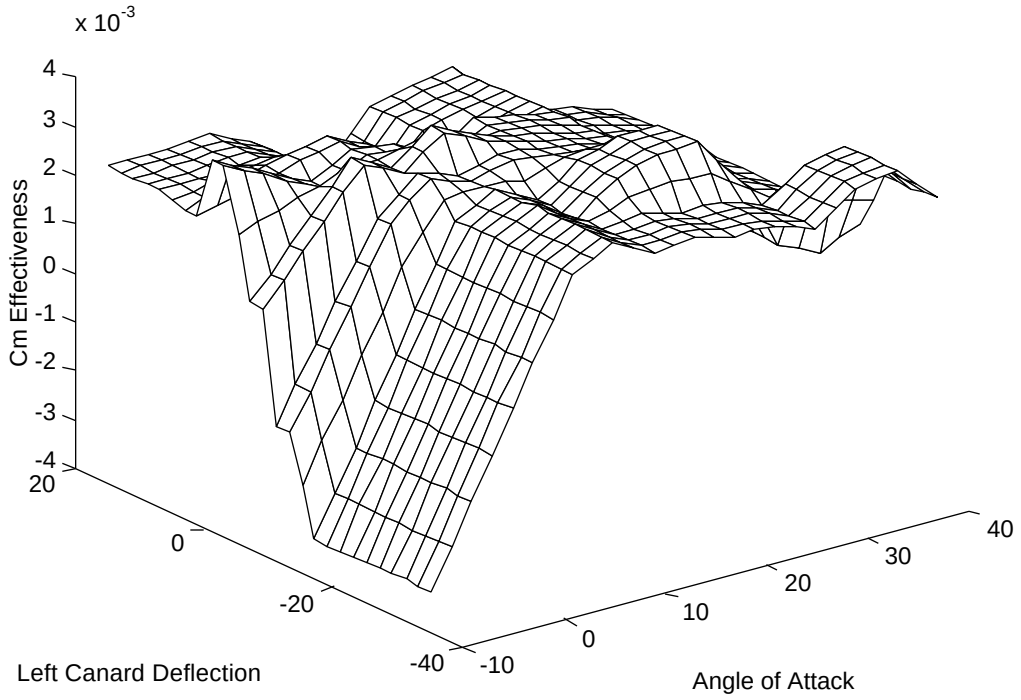

Figure 2.8 Cm Effectiveness ($\partial Cm/\partial \delta$) expressed as a Function $F(\alpha,\delta)$
This plot was generated using Matlab® and the FORTRAN "Sweep Data" utility linked with the F-15 ACTIVE aerodynamic database and the derivative extraction flag set to TRUE. All sweep ranges and reference conditions are identical to those in figure 2.7.

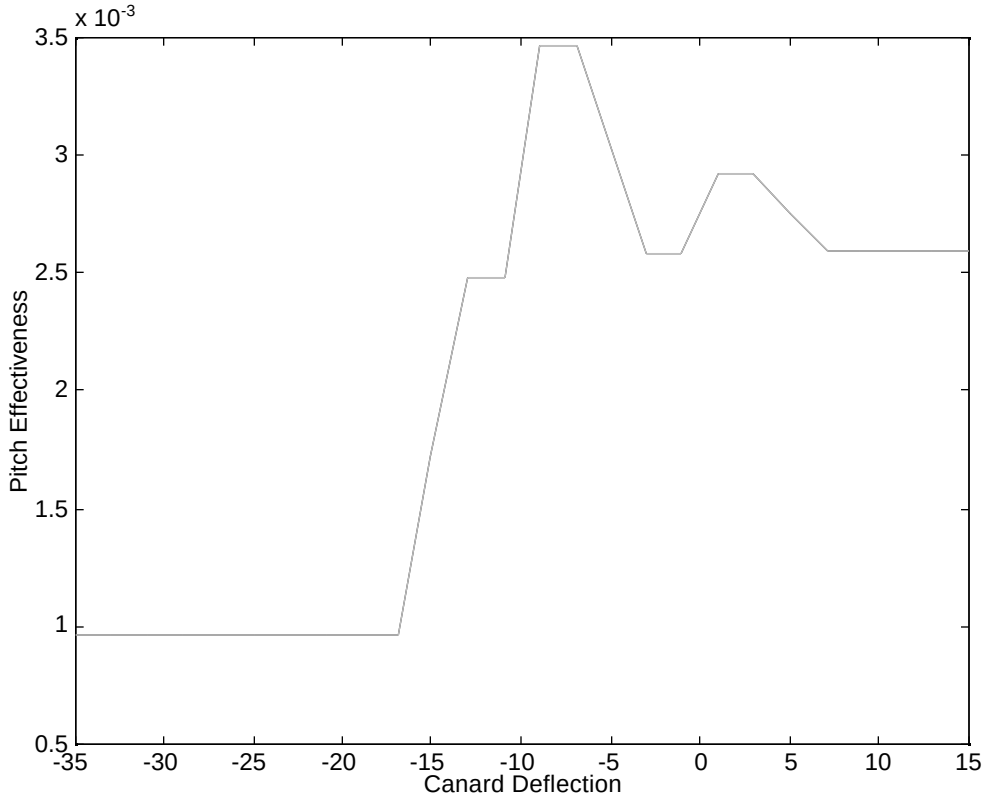

Figure 2.9 Cm Effectiveness ($\partial Cm/\partial \delta$) expressed as a Function $F(\delta)$, $\alpha = 0$
This plot shows the results obtained from using the affine interpolation procedure (dashed line) along with the $\alpha = 0$ slice from figure 2.8 (dotted line). Note that both methods produce identical results.

CHAPTER 3.

Control Allocation with Rate Limiting

This chapter provides background information about the Control Allocation with Rate Limiting (CARL) algorithms that are currently implemented. The name is derived from the fact that these algorithms account for deflection limits, (which were handled in the previous Direct Control Allocation methods), as well as individual surface deflection rate limits. Recall that direct allocation methods produced commanded moments using the most efficient combination of control deflections in the sense that their maximum moment-generating capabilities were utilized, without violating the control position constraints. Likewise, the rate limiting algorithms utilize the controls' maximum abilities in generating moment rates to find an optimal vector of control deflection rates, without violating the respective limits.

To motivate a discussion on these algorithms, the topic of Pilot Induced Oscillations (PIO) is cited. PIO's have been attributed to recent aircraft accidents like the F-22 and the Gripen, both occurring in 1992[9,10]. In fact, there have been postulated theories which suggest that certain types of PIO's may be caused by the non-linearities associated with rate limiting[11,12]. Unlike the direct control allocation schemes that calculate commanded positions based on some commanded moment, the Control Allocation with Rate Limiting algorithms presented here guarantee not to exceed the controls' rate capabilities so long as the control law commands attainable moments. This fact offers a significant benefit to modern control law integration since one of the possible culprits of PIO's can be eliminated. The basic theory behind Control Allocation with Rate Limiting, along with its advantages and disadvantages, will be presented next.

3.1 Early Control Allocation Experiments

The first control allocation algorithms developed were limited by the assumptions of linear control effectiveness and constant constraints. They were also considered primarily as research tools since the methods used to generate the Attainable Moment Subset (AMS) were quite complex and time consuming. Although the theory behind these early tools is beyond the scope of this thesis, it should be pointed out that after developing the more efficient facet-generating algorithms described in references 2 and 3, the implementation of direct control allocation methods could be considered for simulation and real-time applications. Reference 6 describes the first simulator implementation of a direct control allocation algorithm using an F-18 Hornet batch simulation. The version of control allocation used was rather simplistic when compared to the current implementations, yet it still demonstrated the fundamental aspects involved. That is, for a given desired moment vector $\mathbf{m}_d$, a vector of allocated control deflections $\mathbf{u}_a$ can be found such that,

$$\mathbf{m}_d = \mathbf{B}\mathbf{u}_a$$

where $\mathbf{u}_a$ lies in the bounds: $\mathbf{u}_{min} \le \mathbf{u}_a \le \mathbf{u}_{max}$ (3.1)

and the 3 by m $\mathbf{B}$ matrix represents the m ($m > 3$) controls' effectiveness on the 3 aircraft moments.

$$B_{i,j} = \frac{\partial Cm_i}{\partial u_j} \tag{3.2}$$

These early algorithms provided aircraft trajectories very similar to those resulting from the original F-18 control mixing logic, yet, there were some aspects that were somewhat unrealistic. First, the existing F-18 control laws did not compute desired moments, but commanded surface deflections directly. Therefore, a deallocation/reallocation scheme had to be implemented so that the surface deflections provided by the original control laws could be substituted into Eq. 3.1 to give the desired moments, which were then used as inputs to the control allocation software. Second, the control effectiveness data were assumed to be linear across the whole range of control deflections. Although this assumption proved to be a fairly accurate approximation for moments, it prohibited the use of more non-linear effects

(eg. drag), in the control allocation procedure. In other words, this restriction implied that controls were treated as moment generators only, having no effects on aircraft forces. Unfortunately, there may be situations (such as cruising flight), when the drag produced by controls is just as important as the moments that they produce. A final limitation to these methods was the fact that even though the controls' position constraints were accounted for, there was no logic to guarantee that actuator rate limits would not be violated. For reasons mentioned previously, this drawback could become a serious problem for the pilot.

The first problem mentioned could easily be overcome by implementing a different type of control law. Dynamic inversion and Model Following control laws offer a convenient solution since moment commands are easily extracted from their formulations. The limitations of linear effectiveness data and the possibilities of exceeding actuator rate limits were specific to the control allocation algorithms however, and led to the research and development of the rate limiting allocation methods such as CARL.

3.2 Expansion Into The Discrete Time Domain

Equation 3.1 describes the basic operations behind the control allocation algorithms. Given a commanded (or desired) moment vector, a control deflection vector is found which lies within some prescribed minimum and maximum deflection constraints, and produces the desired moment. Through deductive reasoning, it can also be said that given a desired moment rate, the appropriate algorithms could produce a commanded control deflection rate subject to some minimum and maximum rate constraints. An algorithm to achieve this type of allocation procedure is derived as follows:

Differentiate Eq. 3.1 with respect to time. It is assumed that the time rate of change of $\mathbf{B}$ is negligible when compared to the control deflection rates so that:

$$\begin{aligned} \dot{\mathbf{m}}_d &= \mathbf{B}\dot{\mathbf{u}}_a \\ -\dot{\mathbf{u}}_{min} \le \dot{\mathbf{u}}_a &\le \dot{\mathbf{u}}_{max} \end{aligned} \tag{3.3}$$

The minimum and maximum rate constraints prescribed here denote the rate limits of the

controls as they move towards their respective minimum and maximum position limits. Note that while the relations established in Eq. 3.3 describe a moment rate allocation scheme, they neglect the position limits associated with the classic direct allocation methods. However, since modern aircraft control systems are implemented on digital computers in discrete time, it does not make sense to implement this type of algorithm directly. Equation 3.3 can be discretized by approximating the time derivative with a backwards difference equation for the current frame k:

$$\frac{\mathbf{m}_k - \mathbf{m}_{k-1}}{\Delta t} = \mathbf{B}\frac{\mathbf{u}_k - \mathbf{u}_{k-1}}{\Delta t} \tag{3.4}$$

Now, the Δt can be dropped from both sides of the equation, and by using the notion of the first difference[13], Eq. 3.4 becomes:

$$\mathbf{m}_k = \mathbf{B}\ \mathbf{u}_k \tag{3.5}$$

The bounding constraints require that $\mathbf{u}_{min} \leq \mathbf{u} \leq \mathbf{u}_{max}$ (k subscripts dropped).

3.3 Benefits of Control Allocation with Rate Limiting

The relation in Eq. 3.5 is very powerful because it allows the inclusion of both rate limits and position limits as constraints (whichever is the most restrictive). This consideration is accomplished by specifying as the constraints the amount the control surface can move in one frame, determined by either its rate capabilities or position limits, leading to the following:

$$\mathbf{u}_{max} = \min[(\dot{\mathbf{u}}_{max}\Delta t),\ (\mathbf{u}_{max} - \mathbf{u}_k)] \tag{3.6a}$$

$$\mathbf{u}_{min} = \max[-(\dot{\mathbf{u}}_{min}\Delta t),\ (\mathbf{u}_{min} - \mathbf{u}_k)] \tag{3.6b}$$

The first terms in the functions of Eqs. 3.6a and 3.6b contain the continuous time actuator rates multiplied by the sample time, thus giving the amount that the actuators can move during one frame without regard to their positions. Note also that these terms account for the fact that the actuators can move at different rates toward one direction than in the other

direction. As an example, it is often less work on the actuators to move toward a control's free float position (the point at which the hinge moment is zero), than away from it, resulting in a quicker deflection rate for the former situation. The second terms in these functions take into account that a control surface may be close to its physical stops and therefore will not be able to move the distance that its rate capabilities dictate. The current implementation of Control Allocation with Rate Limiting utilizes the discrete time relations just described and is summarized below for a given sample frame k.

1.) The control law calculates a commanded moment vector $\mathbf{m}_k$. CARL calculates the desired change in moment ($\mathbf{m}$) by finding the difference between the commanded moment and the attained moment from the previous frame. The attained moment is based on the current measured control positions and an assumed global-slope control effectiveness matrix.

2.) CARL extracts the local control effectiveness matrix ($\mathbf{B}$), based on the current measured deflections and states and calculates the "delta" constraints according to Eqs. 3.6a and 3.6b.

3.) CARL calculates the "delta" control vector ($\mathbf{u}_k$) such that the controls' rate limits and position limits at the end of frame k are not violated.

Since CARL allocates a change in control deflections at every sample frame, the notion of a "zero deflection" origin in control space can be arbitrarily defined. For this reason, the origin is shifted to the current control positions for every sample frame. The control effectiveness matrix can then be calculated by linearizing about the new reference point. Thus, while the control effectiveness data is still treated as linear, it is done so on a much smaller local-coordinate scale, resulting in better accuracy. Furthermore, the fact that the control effectiveness data can be extracted for the current control positions at each sample frame allows the inclusion of non-linear effects which often vary as the control positions change. The idea of updating the effectiveness matrix at each frame to account for the non-linearities due to controls was demonstrated in Figure 2.3.

3.4 The Control Wind-Up Problem

Unfortunately, the Control Allocation with Rate Limiting algorithm introduces a rather serious problem. Recall from Eq. 3.5 and the 3 steps previously highlighted in section 3.3, that CARL only commands changes in controls based on desired changes in moments. The commanded positions for each sample frame can then be extracted from the definition of the first difference and the previous positions according to:

$$\mathbf{u}_k = \mathbf{u}_{k-1} + \quad \mathbf{u}_k \tag{3.7}$$

so that the control positions at some time $t = n\Delta t$, are the summation of all of the commanded changes in controls for each frame:

$$\mathbf{u}(t) = \mathbf{u}_n = \mathbf{u}_0 + \sum_{k=1}^{n} \quad \mathbf{u}_k \tag{3.8a}$$

Assume that the initial commanded moment vector is zero so that the resulting initial control positions are zero. Equation 3.8a can then be written:

$$\mathbf{u}_n = \sum_{k=1}^{n} \quad \mathbf{u}_k \tag{3.8b}$$

Equation 3.8b reveals an interesting problem. Consider some maneuver that begins at the origin in moment space $\mathbf{m}_0$ (requiring zero control deflections $\mathbf{u}_0$), and moves along some path P1 to another point in moment space $\mathbf{m}_1$, then returns to $\mathbf{m}_0$ along some other path P2. The change in control deflections for each frame will be a function of the current desired change in moment, and the control positions at time t will be the summation of all the previously allocated changes in controls. As a result, if the two paths in moment space P1 and P2 are not identical, it is sufficient to conclude that the final control configuration at the end of the maneuver will generally not return to $\mathbf{u}_0$.

On the other hand, identical paths in moment space do not always guarantee that the controls will return to their initial configuration. In some instances, a constraint may be reached (either rate saturation or position saturation), causing the two paths in control space

to differ. In situations such as this, (even with identical paths in moment space), the problem still occurs. Obviously, the non-zero control deflections are arranged so that the total moment contribution is zero, yet this problem (best described as control wind-up), is undesirable since the controls may be producing unnecessary drag. In addition, the potential exists for some controls to end up close to their limits, which may reduce the maneuvering capabilities of the aircraft at some future time.

The three cases mentioned are demonstrated in the following examples based on the F-15 ACTIVE with a true airspeed of 400 ft/sec, an angle of attack of 8 deg, and an altitude of 10000 ft. Figure 3.1 shows the commanded moments used in these examples. The top plot represents a maneuver which moves from the origin in moment space to some other point, and then returns to zero along the same path, indicated by the symmetry about the t = 10 sec. point. The bottom plot represents a time-asymmetric maneuver which begins and ends at the origin in moment space, but takes a different path going out than coming back in. The nature of these curves is not important since they do not represent any real maneuver. However, the resulting control time histories for all three examples produced identical moments given in Figure 3.1.

Example 1 demonstrates the ideal situation in which the two paths in moment space coincide, with no saturation of controls. The resulting control deflections are shown in Figure 3.2, showing that for this case, the controls are commanded back to their origin.

For a second example, the same symmetric maneuver is used, but the throttle setting has been multiplied by a factor of 2 so that the pitch thrust vectoring nozzles become position saturated during the maneuver. The time histories are plotted in Figure 3.3. Note that although the two paths in moment space coincide, the paths in control space do not because of the position saturation in the thrust vectoring nozzles. As a result, some slight control wind-up occurs and the control deflections do not return to zero.

As a final example, the time-asymmetric maneuver is performed whose commanded moments match those shown in the bottom plot of Figure 3.1. The asymmetric nature of this plot is analogous to saying that the two paths in moment space differ. Therefore, according to the relations described earlier, some control wind-up will occur. The time his-

tories of the control deflections for this maneuver are plotted in Figure 3.4, thus proving this hypothesis.

This chapter has discussed the benefits gained by using the Control Allocation with Rate Limiting algorithms as well as their limitations. The local frame-wise calculations of control effectiveness data allows for the inclusion of non-linear forces such as control-induced drag, and increases the overall accuracy of the attainable moments. In addition, the abilities to include rate limit capabilities and position limits in the control constraints by allocating discrete “delta” controls provide obvious advantages to other control allocation schemes, but the inherent control wind-up problems produce undesired side effects. These problems are alleviated by using the redundant nature of the controls to continuously drive their deflections toward some desired configuration. This technique, known as control restoring, is documented in Chapter 4.

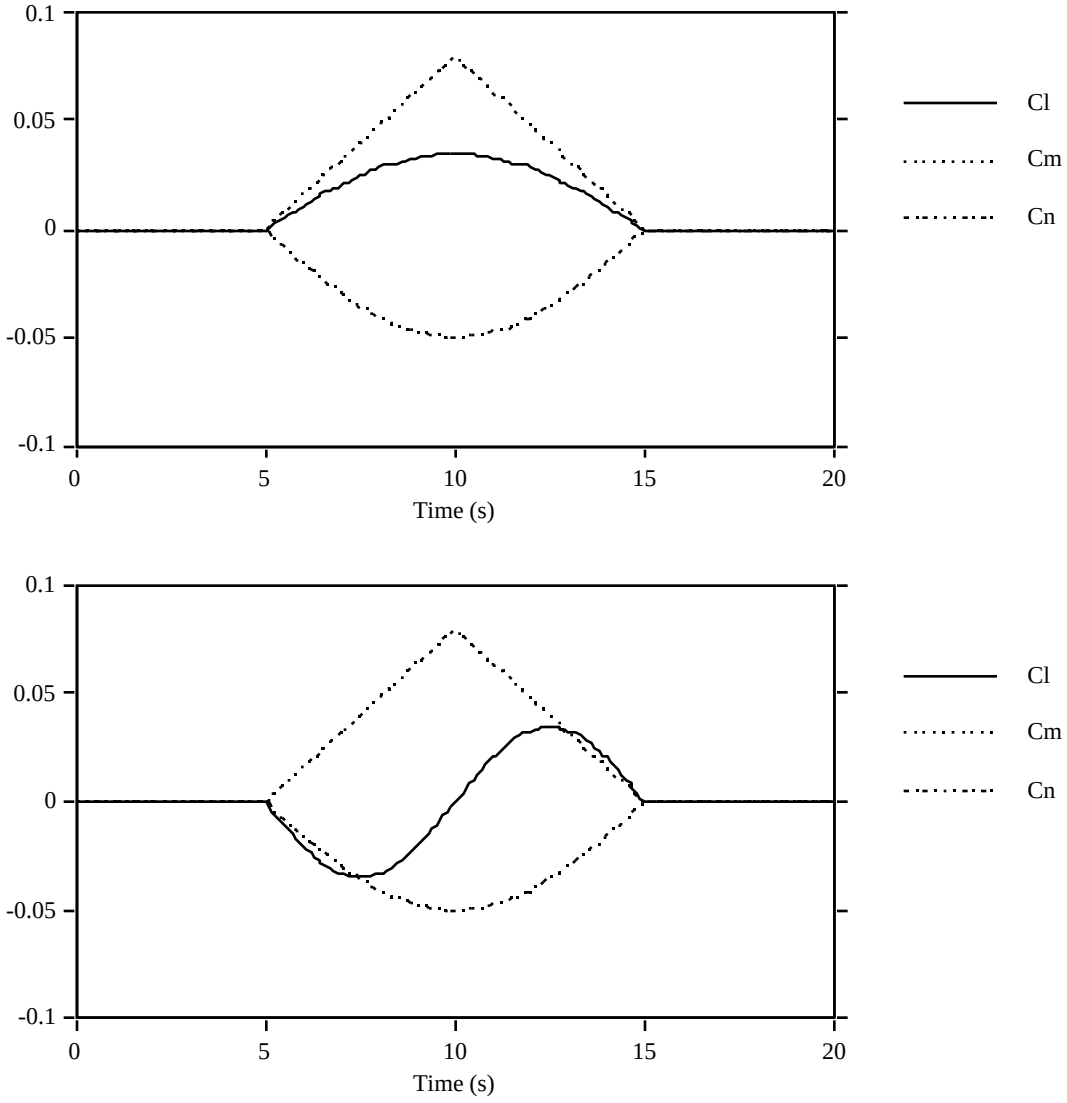

Figure 3.1 Time Histories of Commanded Moments
These two plots represent commanded moment time histories for maneuvers which move from the origin in moment space to some non-zero point and back to the origin. The top graph represents a scenario in which the two paths are equal (indicated by the symmetry about the 10 second point), whereas the bottom plot shows a maneuver that takes two different paths.

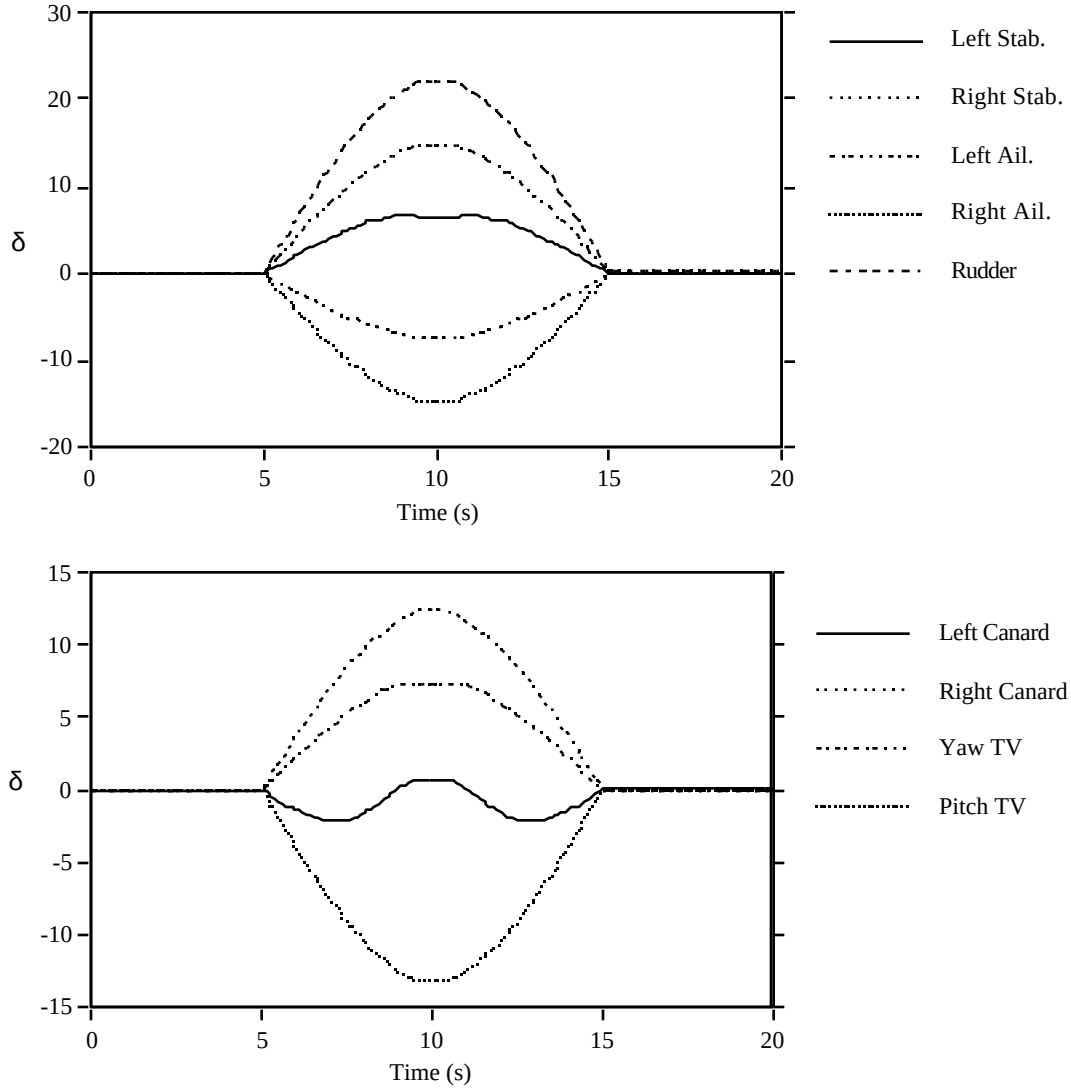

Figure 3.2 Control Time Histories for an "Identical Path" Maneuver (No Saturation)
These plots show the control deflections for the symmetric maneuver shown in Figure 3.1. Since the two paths in moment space are identical, and because no controls are saturated, the commanded deflections return to their starting positions after the maneuver.

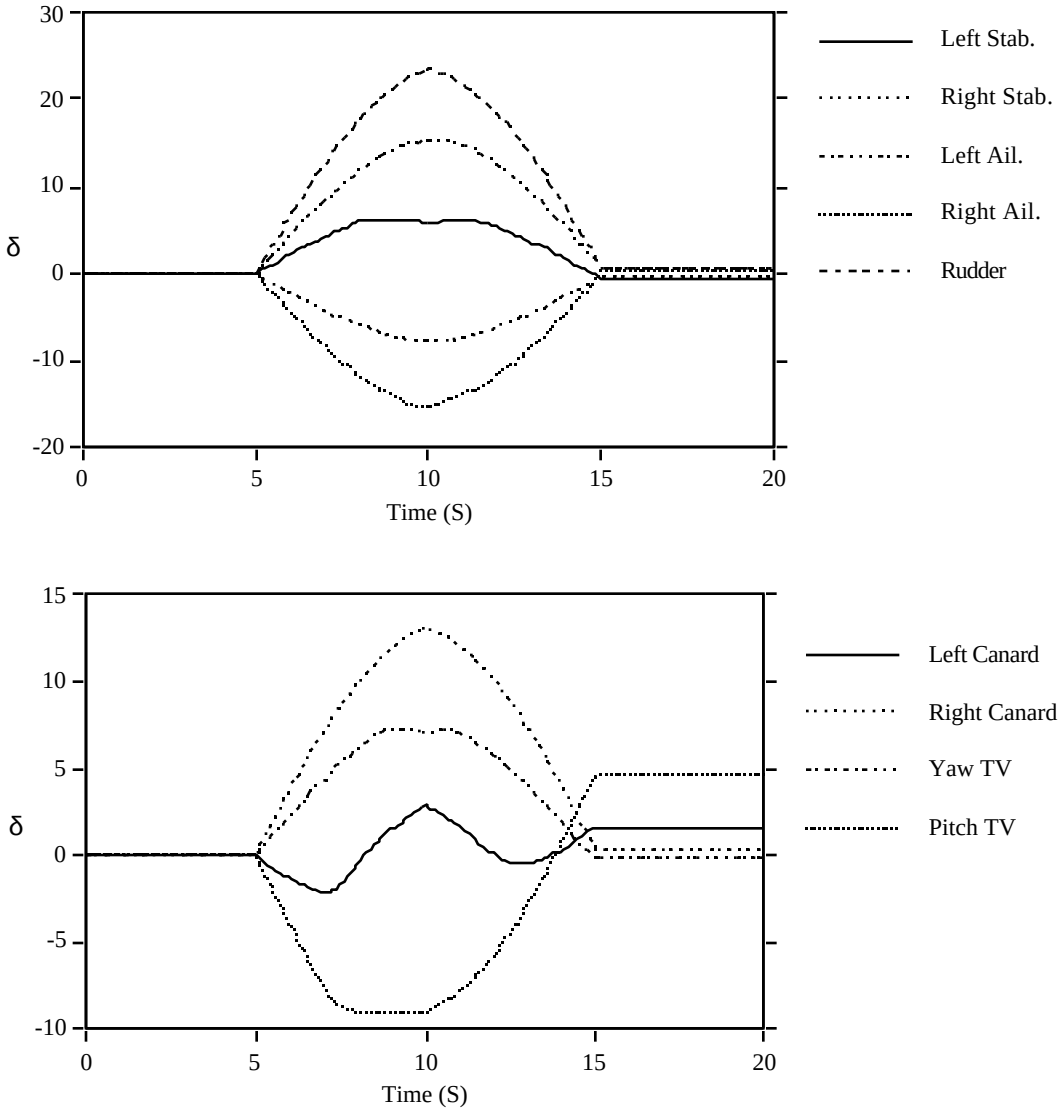

Figure 3.3 Control Time Histories for an "Identical Path" Maneuver (With Saturation)
These plots show the control deflections for the symmetric maneuver shown in Figure 3.1. Although the two paths in moment space are identical, the throttle position has been increased for this example so that the pitch thrust vectoring nozzles become position saturated during the maneuver, resulting in non-zero deflections at the end of the maneuver.

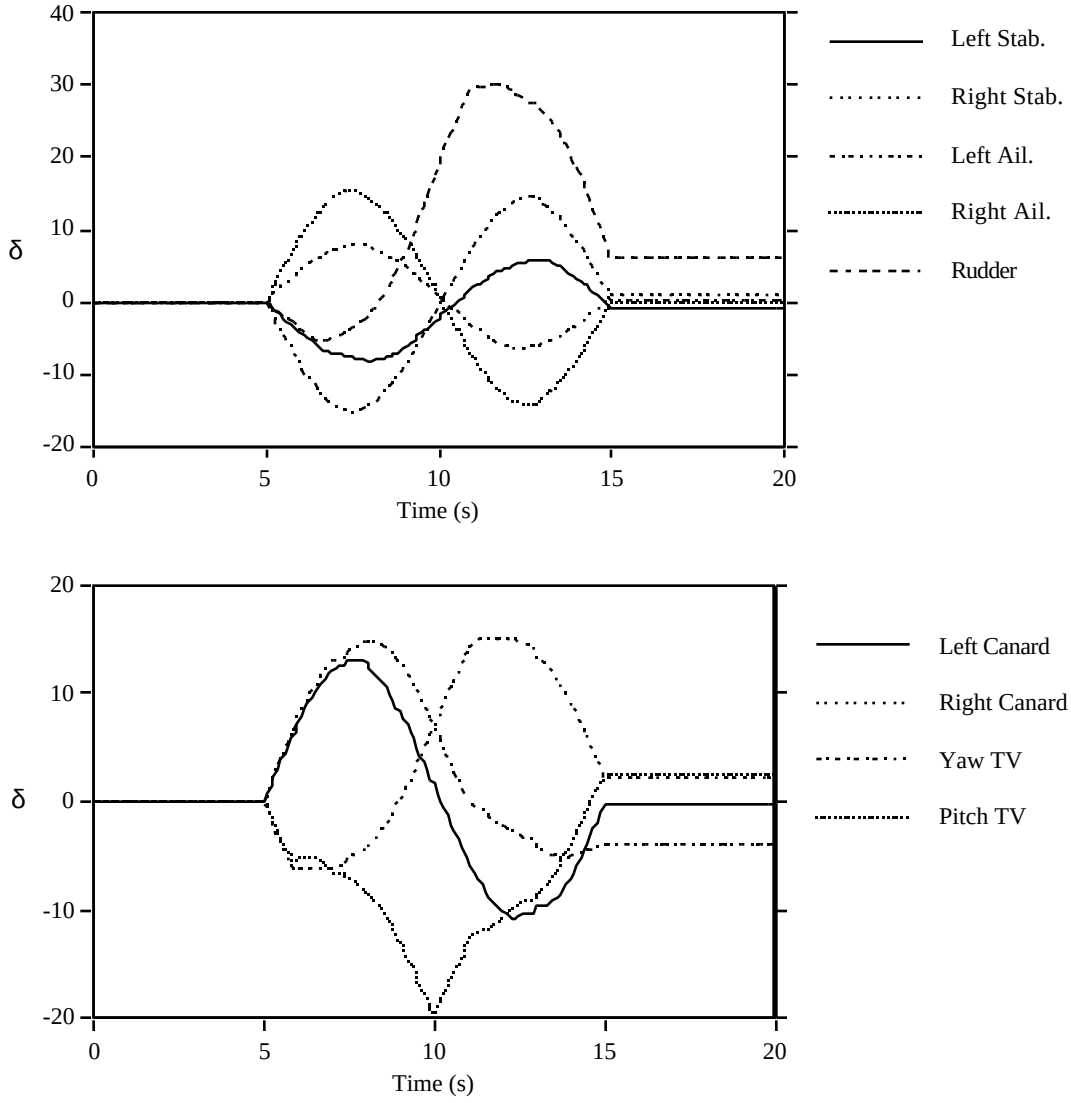

Figure 3.4 Control Time Histories for a Time-Asymmetric Maneuver

These plots show the control deflections for the second maneuver shown in Figure 3.1. Since the two paths in moment space are not equal, the commanded control deflections do not return to their original starting positions.

CHAPTER 4.

Control Restoring Algorithms

In Chapter 3, a control allocation scheme was presented which operated in a discrete time domain by allocating changes in controls based on some desired change in moments for every sample frame. This algorithm proved to have significant advantages over other allocation methods because of its abilities to include non-linear control effectiveness data and to allocate control deflections while avoiding saturation of both control rate and position limits. Yet, due to the discrete nature of the algorithm and path dependencies of the control time histories on the moment time histories, an undesirable control wind-up problem could occur over time, resulting in non-zero control deflections for zero commanded moments.

The notion of zero control positions for zero commanded moments is somewhat arbitrary since the origin can be defined at any desired deflection without affecting the discrete control allocation problem. What is desired, however, is a method which drives the controls to a more suitable control configuration. In other words, if after some maneuver, a control surface has become position saturated due to the wind-up problems of Control Allocation with Rate Limiting, it would be advantageous to restore the control to some desired position between its minimum and maximum limits. Of course, if the commanded moments required the control to be saturated, then the restoring process would affect the behavior of the aircraft and the pilot would have to compensate for the error introduced by the restoring provided. Therefore, control restoring should be considered a minor priority task, and should be used only when there are sufficient moment generating capabilities and control rate capabilities to allow control reconfiguration without affecting the attained moments.

Fortunately, the control allocation algorithms presented herein are designed for situations in which there are redundant controls, (ie. more controls than degrees of freedom). In mathematical terms, the control redundancy produces an under-determined system of

equations for the *m* controls and 3 moments. When linearized, the resulting system then has an infinite number of solutions for any given vector of moments, providing that the control effectiveness matrix has at least one non-singular 3 by 3 partition. From a control allocation perspective, this statement is not entirely correct. Recall that the constraints define the boundary of the Attainable Moment Subset (AMS), which represents moments for which there is only one solution[2]. For moments on the interior of the AMS however, there can be an infinite number of valid control deflection vectors. These can be found by obtaining one solution, and then adding any vector of controls which lie in the null space of the control effectiveness matrix without violating any constraints. Therefore, when the desired moments lie on the inside of the AMS, (indicating that there is some rate and moment generating capability remaining after the desired moment rates have been accounted for), the null space of the control effectiveness matrix can be utilized to drive the controls toward any desired configuration in an effort to alleviate the wind-up problem. It should be noted that because of the infinite possibilities of valid control solutions, there can be an infinite number of restoring methods, and if one is not careful, the restored control deflections may not be any better than the original deflections. This chapter will discuss some of the more practical restoring methods and will present the necessary algorithms needed in obtaining such results.

4.1 Minimum-Norm Restoring: (The Null Space Projection Method)

Recall that in Chapter 3, the wind-up problem was unacceptable because it had the potential of driving the controls close to their position limits, thus reducing available maneuvering capabilities at some future time. With this problem in mind, a procedure is desired which allocates controls such that they remain as close to their zero deflection positions as is possible. This objective is achieved by requiring the 2-norm of the controls to be a minimum, and is accomplished by utilizing the right pseudo-inverse[14] of the control effectiveness matrix defined as:

$$\mathbf{B} = \mathbf{B}^T\left[\mathbf{B}\ \mathbf{B}^T\right]^{-1} \tag{4.1}$$

By using this notion of the pseudo-inverse, the system of equations given by

$$\mathbf{B}\ \mathbf{u} = \mathbf{m} \tag{4.2a}$$

can be solved for **u** according to:

$$\mathbf{u} = \mathbf{B}\quad \mathbf{m} \tag{4.2b}$$

where the vector **u** represents a solution to Eq. 4.2a whose norm is the smallest possible value. The idea of using the pseudo-inverse solution to Eq. 4.2a provides a control allocation scheme in its own. However, an algorithm of this type would not be as efficient as direct allocation since it would not guarantee the use of the controls' maximum moment generating capabilities while avoiding control saturation[4]. Therefore, an algorithm is desired which first allocates controls based on the direct allocation theory described in Chapter 3 to achieve the desired moments, and uses any remaining rate capability to continuously drive the controls toward their minimum-norm solution without affecting the attained moments obtained from direct allocation. This type of restoring is outlined as follows:

1.) Based on the current moment commands $\mathbf{m}_k$ and the global (slope at the origin) control effectiveness matrix **B**, find the minimum-norm solution of controls $\mathbf{u}_p$ based on the pseudo-inverse in Eq.. 4.2b.

2.) Calculate the difference in control deflections from the pseudo-inverse solution and the control positions obtained in the previous frame according to:

$$\mathbf{u}_p = \mathbf{u}_p - \mathbf{u}_{k-1} \tag{4.3}$$

Note that substituting $\mathbf{u}_p$ into Eq. 4.2a produces the current change in desired moment $\mathbf{m}_k$ used by the control allocation algorithms.

3.) Find the difference between the former change in controls and the change in controls given by the control allocation algorithms:

$$\mathbf{u}' = \mathbf{u}_p - \mathbf{u}_k \tag{4.4}$$

Thus, by the definitions of $\mathbf{u}_p$ and $\mathbf{u}_k$, and by assuming that the control effec-

tiveness matrix is constant with respect to control position, the difference $\mathbf{u}'$ substituted back into Eq. 4.2a, results in a vector of zeros, and therefore lies in the null space of $\mathbf{B}$.

4.) The pseudo-inverse equation used in step 1 has no knowledge of the control constraints and will generally produce results which, when combined with the allocated control vector in step 3, give a $\mathbf{u}'$ vector which violates either a rate limit or position constraint for one or more controls. Therefore, it is important to scale this vector as necessary so that no individual control surface violates its constraint. This scaling step can be easily accomplished by shifting the origin in control space after the allocated control vector has been found using:

$$\mathbf{u}'_{min} = \mathbf{u}_{min} - \mathbf{u}_k \tag{4.5a}$$

$$\mathbf{u}'_{max} = \mathbf{u}_{max} - \mathbf{u}_k \tag{4.5b}$$

and applying a variable scale factor K to the $\mathbf{u}'$ vector so that none of the shifted constraints are violated. The method for finding K involves inspecting each restored control in the restoring vector $\mathbf{u}'$, and proceeding as follows:

i.) for every u' greater than its shifted maximum constraint u'_{max}, find the ratio:

$$r_i = \frac{u'_{max}}{u'} \tag{4.6a}$$

ii.) for every u' less than its shifted minimum constraint u'_{min}, find the ratio:

$$r_i = \frac{u'_{min}}{u'} \tag{4.6b}$$

iii.) The scaling factor K, is then the minimum of these evaluated ratios:

$$K = \min(r_i) \tag{4.6c}$$

5.) As a final step, the commanded control vector for the current frame is built up

according to:

$$\mathbf{u}_k = \mathbf{u}_{k-1} + \quad \mathbf{u}_k + K \quad \mathbf{u}' \tag{4.7}$$

Equation 4.7 will be called the restoring equation. Two key aspects of this equation should be evident. First, by ignoring the last term on the right side of the equality (the restoring term), the equation reduces to the same form as Eq. 3.7 (basic control allocation with rate limiting). Second, the restoring term takes into account the difference between the allocated change in controls and the required change needed to approach the pseudo-inverse solution, so that during maneuvering flight, the controls are continuously driven toward their minimum-norm configurations, provided that the scale factor K, never becomes zero. In static flight, the moments are not changing, and the allocated changes in controls will be zero. In this case, the restoring term represents the difference between the minimum-norm solution and the current control configuration. The controls will then be commanded to move until the restoring term is zero, implying that the controls have reached their minimum-norm configuration.

While this restoring technique initially seemed to fix the control wind-up problem, it was later discovered that the algorithm would fail under certain situations. Recall that the pseudo-inverse solution calculated in step 1 uses the control effectiveness matrix linearized about the zero control positions (global slope data), whereas the allocated control vector uses local control effectiveness data as a function of current control position. In general, if the non-linear effects with respect to the controls are significant, then the restoring vector calculated in step 3 will not lie in the null space of the local control effectiveness matrix and the restoring process will result in errors between the commanded and attained moments. The assumption of constant control effectiveness with respect to control position is generally adequate for small deflections, yet, under circumstances requiring large deflections, the underlying theory of the null space projection method breaks down. This problem and the proposed solution will be presented later in this chapter. For now however, the discussion of restoring techniques will continue in the chronological order in which they were implemented.

4.2 Minimum-Drag Restoring

The idea of including control generated forces as well as moments when allocating controls was first introduced in reference 2. Yet, because of the globally linear assumptions inherent in these early algorithms, it did not make sense to include such forces because their effects were significantly more non-linear across the controls' deflection ranges than the moment effects. As an example, the drag increments due to the F-15 ACTIVE left stabilator were plotted in figure 2.5. For that particular flight condition, the control-induced drag was a minimum for a 10 degree trailing edge up deflection, and increased as the control position strayed from this minimum-drag point. By taking these characteristics into consideration, it would be impossible to prescribe a non-zero global slope to account for the drag effectiveness of the left stabilator. However, by using the Control Allocation with Rate Limiting (CARL) algorithms, which make use of locally linearized data about smaller frame-wise deflection limits, and assuming that these frame limits are suitably small, then the non-linear effects can be included since the assumptions of linearity become more valid.

Minimizing control-induced drag is certainly a design consideration when developing control allocation or control mixing algorithms. However, the idea of using minimum-drag control restoring is based on the intuition that such deflections produce a "clean" configuration. Such configurations are normally the case for low to moderate angles of attack and angles of side-slip. But this technique begins to break down as these angles become large in magnitude since the minimum control-induced drag deflections tend to migrate toward their position constraints. In cases such as this, it may be more beneficial to revert to a minimum-norm restoring algorithm instead. An example illustrating this fact will be presented towards the end of this chapter.

By including drag in the control allocation problem, the control effectiveness matrix defined in Chapter 3 gets augmented with a 4th row corresponding to each control's effect on drag. That is,

$$\mathbf{B}_{4,\mathrm{j}} = \frac{\partial C_D}{\partial u_j} \tag{4.8}$$

Thus, the augmented **B** matrix now represents a transformation from control space to the Attainable Objective Subset (AOS) whose coordinates consist of the 3 control-induced moments plus drag. The mathematical problem of finding the AOS geometry is practically the same as in the three-moment problem and its associated Attainable Moment Subset (AMS) described in references 2 and 3. The difference between the two is the increase in the dimensionality of all of the matrices and vectors required to generate the 4-dimensional AOS. The actual allocation of controls however, is slightly different due to the fact that the 4th objective (eg. drag) is not specified explicitly like the three aircraft moments, but is prescribed to be minimized. A procedure for an allocation scheme of this nature with examples was described in detail in reference 5. To motivate this discussion, the direct control allocation problem is revisited.

Direct control allocation, as it pertains to a vector space of objectives, is accomplished by first determining the geometry of the AOS for the given controls' effectiveness and physical constraints, and then calculating which one of the bounding facets the current objective vector points toward. The coordinates of the objective vector's intersection with one of these facets represents the maximum attainable value in that direction in objective space, and can be transformed back to control space to give the unique solution of admissible controls associated with that intersection. These controls are then scaled to achieve the desired objective magnitude.

In the current discussion, the objectives are the changes in the three aircraft moments and the desired change in drag. Define the objective vector as:

$$\mathbf{C} = \left\{ C_l,\ C_m,\ C_n,\ C_D \right\}^{\mathrm{T}} \tag{4.9}$$

If the desired objective vector is specified as $\mathbf{C} = \{0,\ 0,\ 0,\ -1\}^{\mathrm{T}}$, then direct allocation will first find the intersection of this vector with the AOS, and if the scaling step is omitted, will return an allocated change in controls which produces no change in moment coefficients, but produces the greatest reduction in drag coefficient. Recall that the constraints imposed on **u** are the most restrictive of either the controls' rate limits or their position limits. Assuming the former is the more restrictive, then the controls will move at their maximum rate toward their minimum drag configuration.

The vector $\{0, 0, 0, -1\}^T$ certainly demonstrates the minimization capabilities of direct allocation, but it does not satisfy any arbitrary desired moment commands. The vector $\{ C_l, C_m, C_n, -1\}^T$ does not solve the problem either since all four components will have to be scaled to the boundary of the AOS. What is required then is a method that first calculates a $\mathbf{u}$ which yields the desired moment commands $\{ C_l, C_m, C_n\}^T$ for an unspecified C_D, relocates the origin of the AOS to this point, and then allocates a further $\mathbf{u}'$ satisfying the minimization vector $\{0,0,0,-1\}^T$. That is,

1.) Solve the classical three-moment problem from:

$$\mathbf{m}_k = \mathbf{B}_{(3xm)} \quad \mathbf{u}_k; \quad \mathbf{m}_k = \{ C_l, C_m, C_n \}^T \tag{4.10}$$

subject to the constraints $\mathbf{u}_{min} \leq \mathbf{u} \leq \mathbf{u}_{max}$ (k subscripts dropped). Note that this is no different from the CARL algorithm presented in Section 3.2.

2.) Shift the origin in control space to the allocated vector of controls $\mathbf{u}_k$. This step also requires that the constraints be shifted according to:

$$\mathbf{u}'_{min} = \mathbf{u}_{min} - \mathbf{u}_k \tag{4.11a}$$

$$\mathbf{u}'_{max} = \mathbf{u}_{max} - \mathbf{u}_k \tag{4.11b}$$

3.) Solve the 4-objective problem on the boundary of the AOS from:

$$\mathbf{C} = \mathbf{B}_{(4xm)} \quad \mathbf{u}'; \quad \mathbf{C} = \{0,0,0,-1\}^T \tag{4.12}$$

subject to the shifted minimum and maximum constraints defined above.

4.) Finally, apply the restoring equation:

$$\mathbf{u}_k = \mathbf{u}_{k-1} + \mathbf{u}_k + K \mathbf{u}' \tag{4.13}$$

The 4-objective problem solved in step 3 is in fact a direct allocation problem. Therefore, the limits imposed on the controls by either their rate capabilities or position constraints are guaranteed not to be violated. The purpose of the scaling factor in step 4 is to account for the slope reversals in drag effectiveness about the minimum-drag positions, which generally cause the controls to oscillate about such positions. This factor can be set to some value

between 0 and 1, where 0 results in no restoring, and 1 causes the controls to use all of their remaining rate capabilities for the restoring process.

Additional trends to note are that small values of K tend to decrease the amplitudes of the minimum-drag control oscillations at the price of obtaining a slower convergence to the minimum-drag configuration. Second, the algorithm as presented does not purport to converge to a minimum drag position for every frame. An optimization problem of this nature would be quite time consuming and impractical since the minimum-drag configuration for one frame may be obsolete in the next. Instead, it drives the controls toward their minimum-drag deflections at each frame. Only in cases of static moment commands for extended periods of time will it actually achieve a minimum drag configuration.

As mentioned in section 4.1, the null space projection method proved to be a technology demonstration only. Because of an invalid assumption, its use had to be limited to relatively small magnitude deflections and as a result, it was not well suited for any real-time implementation. The minimum-drag restoring described above, although providing the benefits of being able to include non-linear control effectiveness data, utilizes a 4-dimensional direct allocation method which generally requires more computational time. These problems led to the development of more efficient algorithms that have the speed benefits associated with the null space projection method, and the non-linear capabilities of the minimum-drag 4-dimensional allocation scheme. Such methods will be generalized as non-linear restoring techniques and may include non-linear minimum-norm restoring, minimum-drag restoring, or any other restoring objective that can be expressed in terms of current control deflections.

4.3 Non-Linear Restoring Techniques

For the minimum-drag restoring mentioned before, a 4-dimensional direct allocation method was originally employed. Although this algorithm performs exceptionally well, it may be considered overkill for such a minor priority task. That is, during a maneuver, it is most important that the commanded moments be obtained if possible. Direct allocation methods ensure this by using the controls' maximum capabilities in generating moments.

What is not so important however, is how the controls are restored to some desired configuration. For instance, it is not necessary that the controls take advantage of their maximum capabilities in reducing drag to get to some minimum-drag configuration, but only that they tend toward this configuration. Therefore, a restoring algorithm will be used that may not utilize the best combinations of controls to decrease drag, but saves valuable computational time. The resulting algorithm is a hybrid of the null-space projection method and the 4-dimensional allocation method in that it utilizes the pseudo-inverse solution based on the 4-dimensional local slope control effectiveness matrix. It is summarized below:

1.) Solve the classical three-moment problem from:

$$\mathbf{m}_k = \mathbf{B}_{(3xm)} \quad \mathbf{u}_k; \quad \mathbf{m}_k = \{ \quad C_l, \quad C_m, \quad C_n \}^T \tag{4.14}$$

subject to the constraints $\mathbf{u}_{min} \leq \mathbf{u} \leq \mathbf{u}_{max}$ (k subscripts dropped)..

2.) Augment the control effectiveness matrix with a 4th row containing the objective data to minimize.

3.) Solve the 4-objective problem using the pseudo-inverse of the augmented **B** matrix for

$$\mathbf{C} = \mathbf{B}_{(4xm)} \quad \mathbf{u}'; \quad \mathbf{C} = \{ 0,0,0, -1 \}^T \tag{4.15}$$

Note that the structure of the objective vector will result in a solution which lies in the null space of the original 3 by *m* control effectiveness matrix, but moves in the direction for which the objective decreases.

4.) Since the pseudo-inverse solution has no knowledge of control constraints, they must be scaled in the same manner that was done in step 4 of Section 4.1 (K_p).

5.) To decrease control chattering about their minimum-objective configurations, an additional minimization factor must be applied as in step 4 of Section 4.2 (K_m).

6.) Based on the previously obtained control positions $\mathbf{u}_{k-1}$, the allocated control vector $\mathbf{u}_k$, the restoring vector $\mathbf{u}'$, and the two scaling factors K_p (from step 3) and K_m (step 4), restore the controls according to:

$$\mathbf{u}_k = \mathbf{u}_{k-1} + \quad \mathbf{u}_k + K_m \quad K_p \quad \mathbf{u}' \tag{4.16}$$

The procedures for this algorithm remain the same regardless of the restoring objective used. The only difference lies in the type of data used to generate the 4th row of the **B** matrix in step 2. For minimum-drag restoring, this data is the control effectiveness in drag that can be extracted from the aerodynamic database. For minimum-norm restoring, this row can be thought of as the control derivatives of some continuous function whose minimum occurs at zero. In the current implementations, this function is the value of the control positions (converted to radians) raised to the second power:

$$F = \left(\frac{\pi}{180}\, u\right)^2 \tag{4.17a}$$

so that the actual effectiveness data used for each 4th row entry is:

$$\frac{\partial F}{\partial u_i} = \frac{\pi}{90}\, u_i \tag{4.17b}$$

The purpose of the degrees to radians conversion in Eq. 4.17a, is so that the 4th row of the control effectiveness matrix contains entries that are of the same order of magnitude as the other control effectiveness entries. Of course, some implementations may be such that not all of the controls can be expressed as deflections. In cases such as this, a different objective function may have to be defined. The following section will demonstrate the effects of control restoring using this algorithm.

4.4 Effects of Control Restoring

In Chapter 3, a maneuver was demonstrated which originated at the origin in moment space, moved to some other point along a specified path, and returned back to the origin along a different path. Due to the path dependent nature of the control deflections with the moment commands, this maneuver resulted in non-zero control deflections for zero commanded moments. For reference, the control time histories are reproduced here in Figure 4.1.

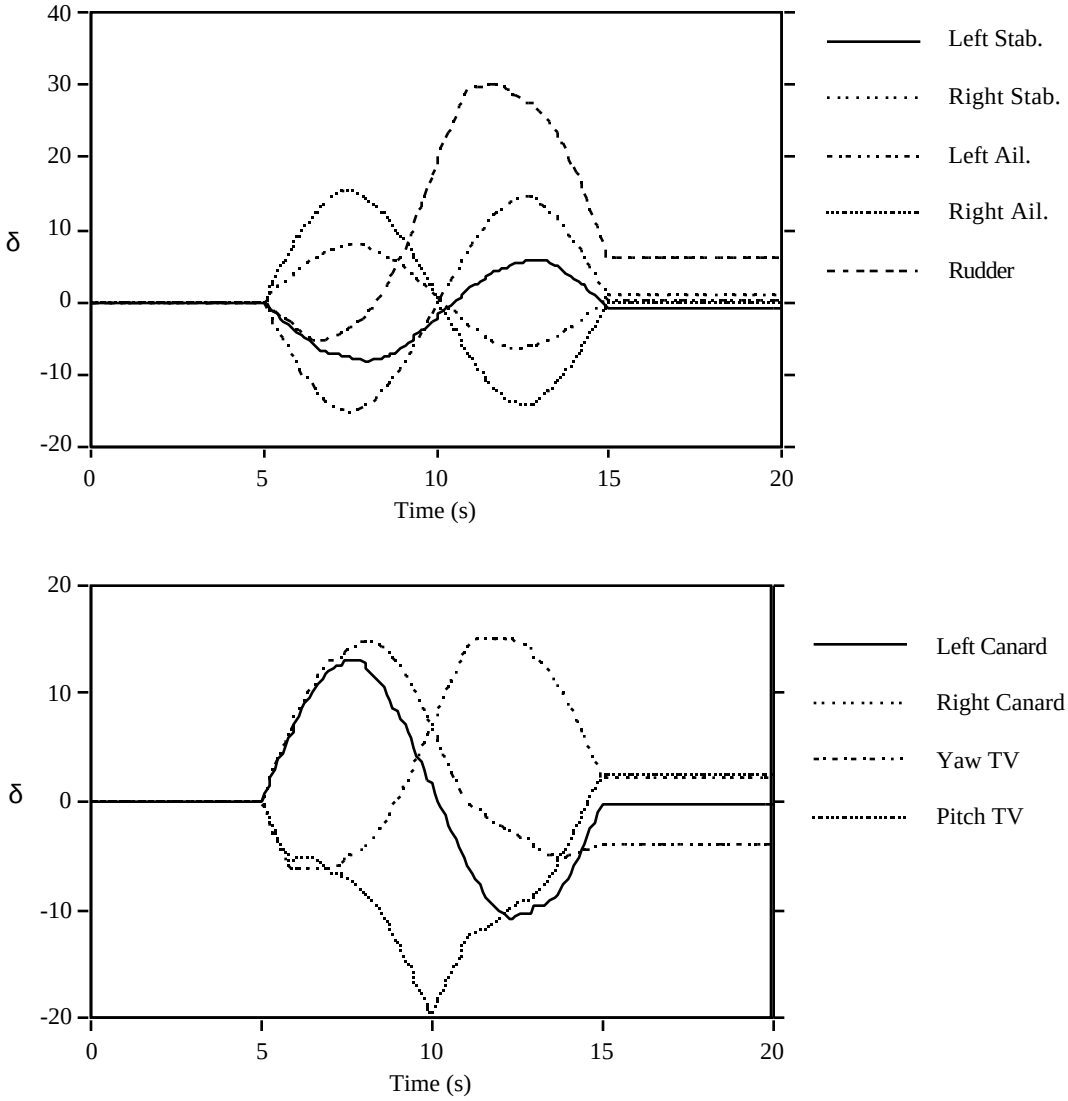

Figure 4.1 Control Histories For a Time-Asymmetric Maneuver (No Restoring)
This figure shows the control deflections for the maneuver depicted in the second plot of Figure 3.1 with no restoring, and demonstrates the control wind-up problem discussed in Chapter 3.

The fact that the controls are non-zero after the maneuver is not entirely undesirable, since the controls could possibly result in some symmetrical configuration. As an example, drooping the ailerons during power approach is not uncommon, even though for this flight regime the required moments are for the most part constant. What is undesirable about the wind-up problem however, is that the controls are not guaranteed to be symmetric for static conditions. In Figure 4.1, the rudders wind up to a 6 degree trailing edge left deflection, whereas the yaw thrust vectoring nozzles remain deflected approximately 4 degrees to the right, most likely to cancel the yaw produced by the non-zero rudder deflections. In addition, there are also some amounts of differential deflection remaining in the canards, stabilators, and ailerons even though the commanded and resulting moment vector is zero.

Figure 4.2 compares the attained moments with the commanded moments for the same time-asymmetric maneuver introduced in Chapter 3, but with minimum-norm restoring (first plot) and minimum-drag restoring (second plot) enabled. The purpose of these plots is to demonstrate the fact that these restoring techniques, while producing significantly different control time histories than the standard CARL algorithms, result in deflections which lie in the null space of the control effectiveness matrix. In other words, the deflections produced by the restoring process do not contribute to the overall moments produced in the previous direct allocation steps. The resulting sets of control time histories for the different restoring methods are shown in Figures 4.3 and 4.4.

Figure 4.3 represents the control deflections for the minimum-norm restoring example. It should be noted that these deflections are generally smaller in magnitude than those with no restoring shown in Figure 4.1. This result is no surprise since the objective function implemented for this scheme is prescribed to have a minimum at zero deflection for all controls. It is also evident that this type of restoring will eventually produce zero control deflections for zero commanded moments.

Figure 4.4 represents the control time histories for minimum-drag restoring. This is one method which does not necessarily command zero deflections for zero moments. However, because the final flight condition for the sample maneuver is symmetric, it commands symmetric deflections. All of the controls which affect the lateral and directional axes only (eg. ailerons, rudder, and yaw thrust vectoring), are restored to their zero positions.

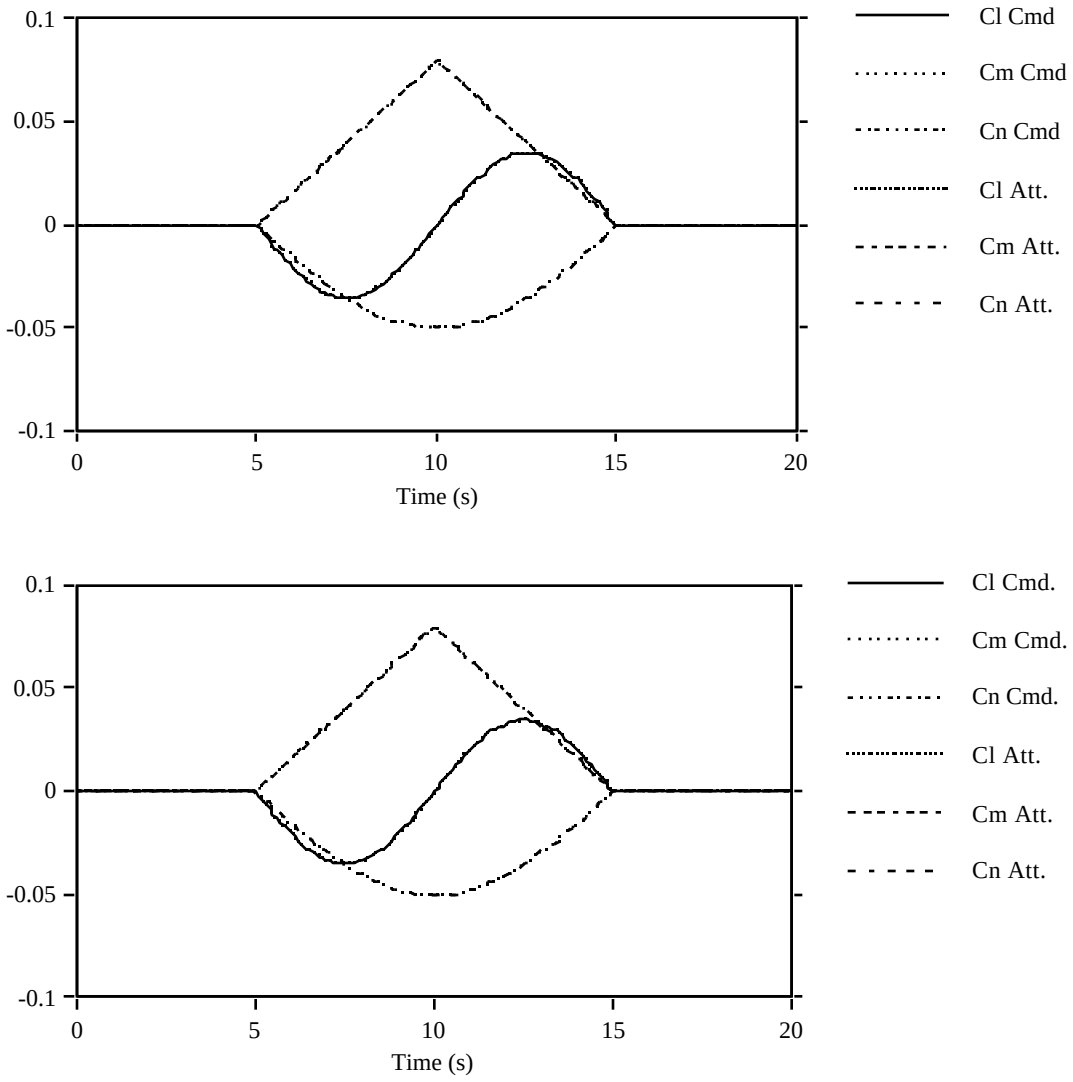

Figure 4.2 Commanded Moments and Attained Moments for Control Allocation with Restoring

These plots compare the commanded moments with the actual attained moments for control allocation with minimum-norm restoring (top), and control allocation with minimum-drag restoring (bottom). The fact that the attained moments coincide with the commanded moments proves the idea that control restoring is performed within the null space of the control effectiveness matrix, so that it has no effect on the control-generated moments.

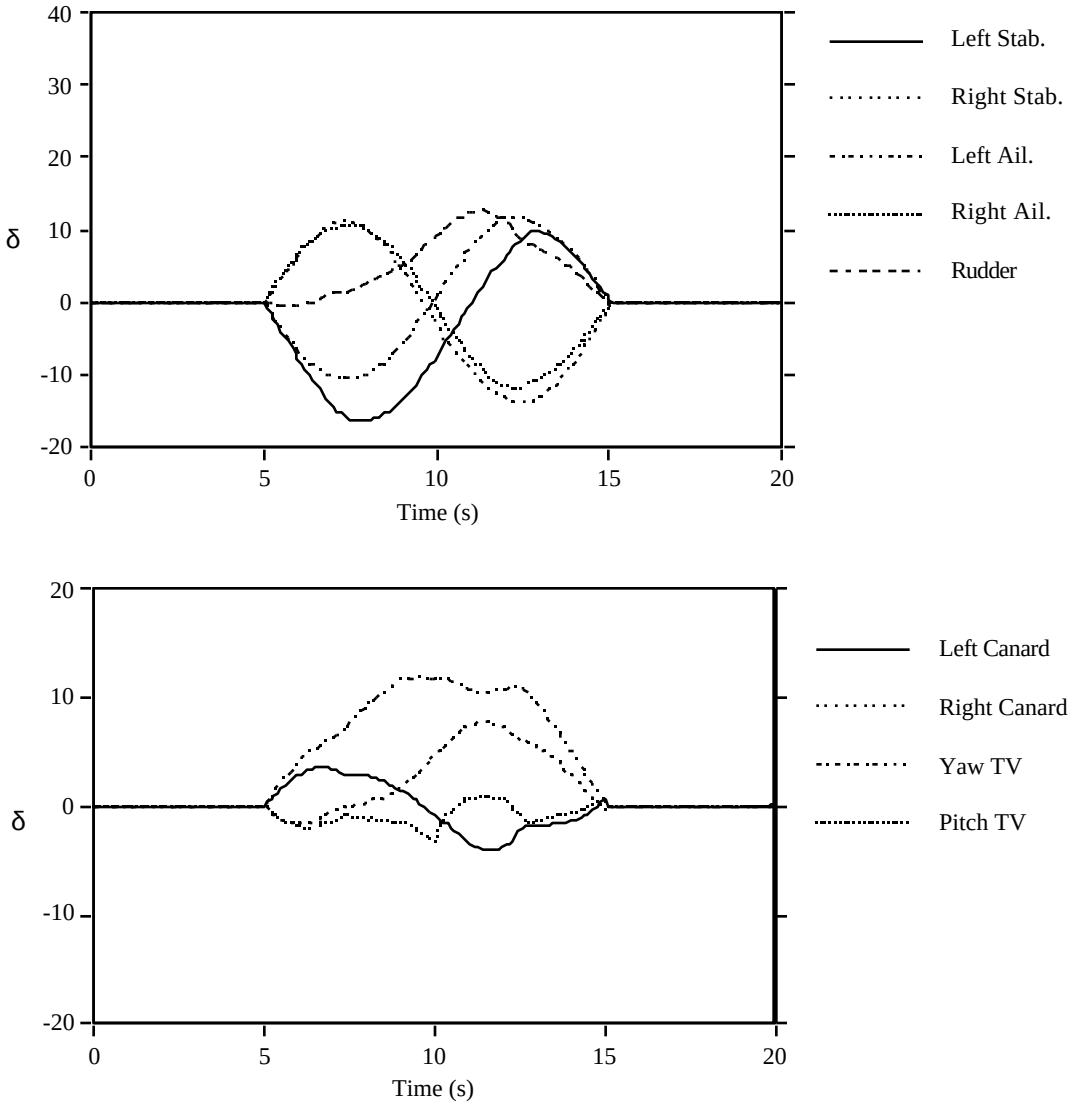

Figure 4.3 Control Histories For a Time-Asymmetric Maneuver (Minimum-Norm Restoring)

This figure shows the control deflections for the maneuver depicted in Figure 4.2 with minimum-norm restoring. Note that the control deflections are generally smaller in magnitude, and that they return to their zero deflections after the maneuver.

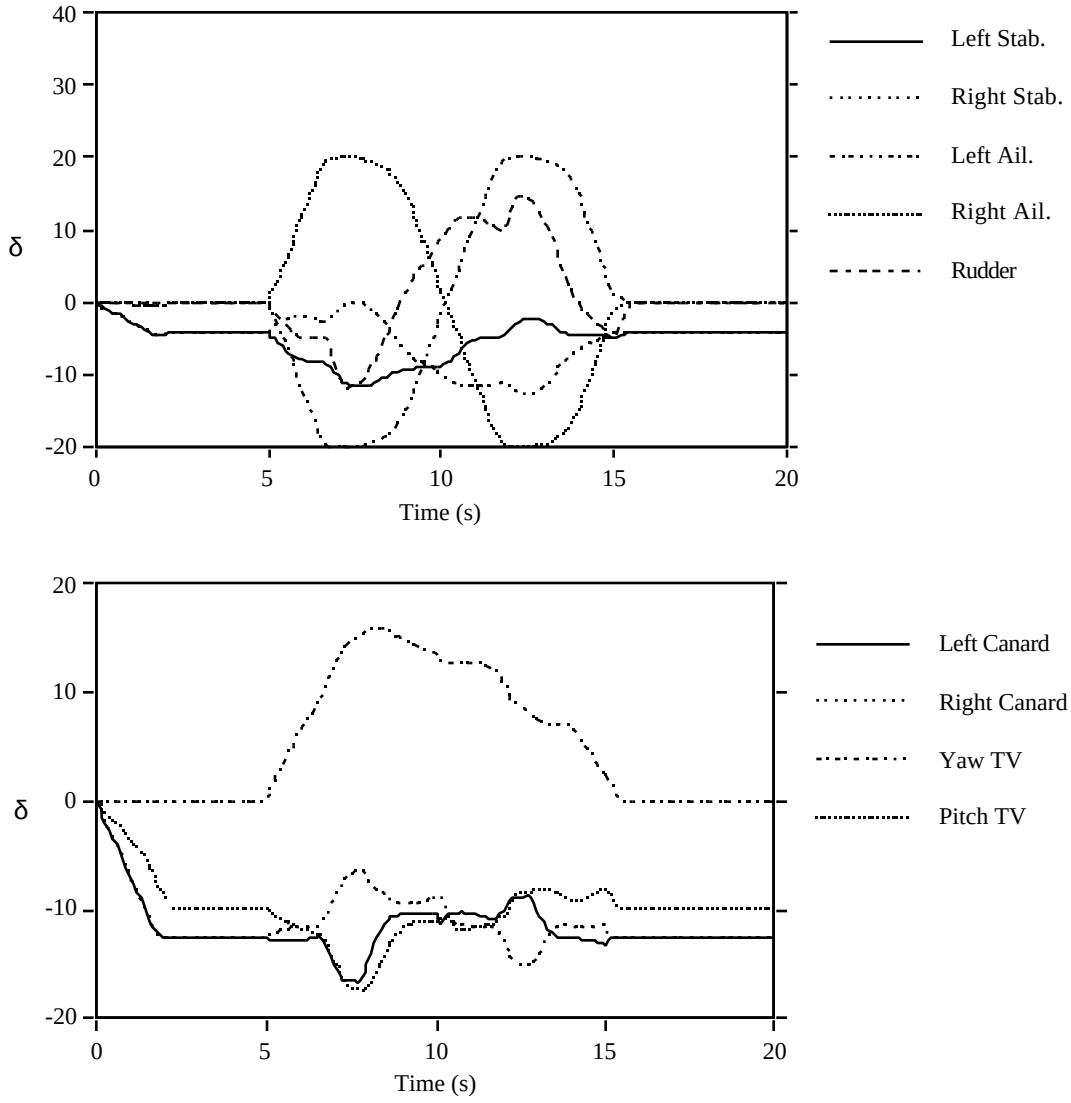

Figure 4.4 Control Histories For a Time-Asymmetric Maneuver (Minimum-Drag Restoring)

This figure shows the control deflections for the maneuver depicted in Figure 4.2 with minimum-drag restoring. In this case, the controls do not necessarily return to their original positions, but return to their minimum-drag configurations for this particular flight condition.

Of course, if the final flight condition was not symmetric (perhaps a steady side-slip condition), then these controls would not be commanded to zero because their minimum drag positions would change. The longitudinal controls (canards, stabilators, and pitch thrust vectoring) are restored to non-zero positions in order to minimize their respective increments in drag, while at the same time, producing control-induced moments which cancel each other out, so that the overall moment contribution is zero.

From the example in Figure 4.4, Minimum-Drag restoring is seen as a method which generally does not command zero controls for zero moments. One should be careful when implementing these types of algorithms since there is always the possibility that the restored controls will not be any more advantageous than the unrestored controls. A fundamental problem with minimum-drag restoring is the fact that at some flight conditions, the minimum control-induced drag configurations may require the controls to be close to their physical limits. This presents a disadvantage to the pilot since it may inhibit the maneuvering capabilities of the aircraft at some future time. As a demonstration, the controls associated with a static flight condition typical of a high angle of attack, level flight, trim condition are shown in Figure 4.5. For this example, the initial conditions were set for a 15 deg. angle of attack, 270 ft/sec velocity and an altitude of 10000 ft. The initial control deflections were set by direct allocation using global slope effectiveness data and physical position constraints for a static control-generated pitching moment coefficient of approximately 0.02. A simulation was then run for 8 sec with minimum-drag restoring enabled. Since the total commanded moment vector remained constant throughout this simulation, CARL did not allocate controls, and the only changes in commanded control deflections were direct results of the restoring process only. From this figure, a rather serious problem with minimum-drag restoring is revealed, in that some of the controls have been driven to their position limits. It is verified in Figure 4.6 that for this particular flight condition, the minimum-drag configuration for the controls requires rather large trailing edge up deflections. In cases such as the ailerons and thrust vectoring nozzles, the minimum actually occurs at the physical position limits.

It is also interesting to note that the slope of the drag effectiveness for the pitch thrust vectoring nozzles remains positive throughout the entire deflection range.

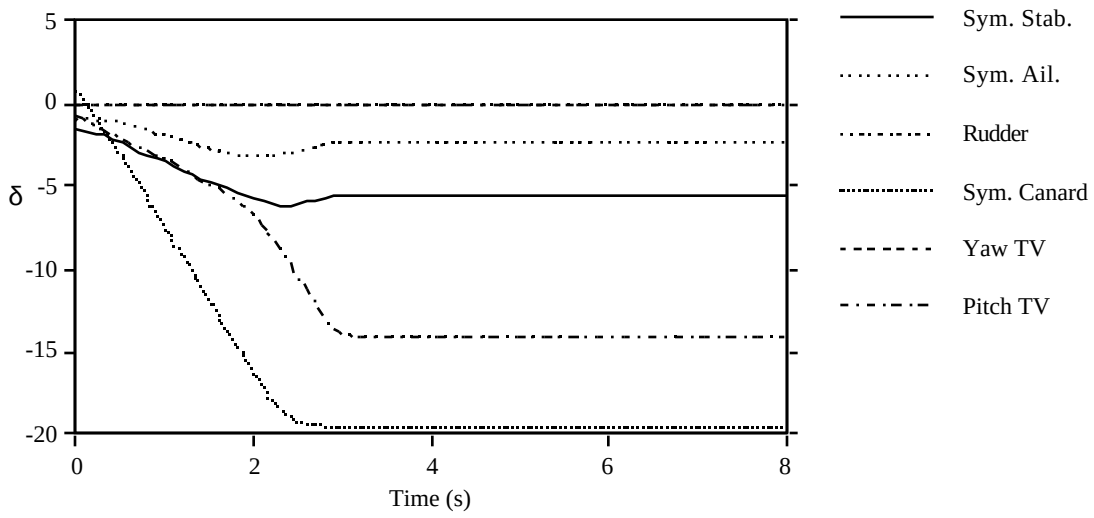

Figure 4.5 Minimum-Drag Restoring for a Static Flight Condition
This figure shows the control deflections for a static flight condition at 15 degrees angle of attack, and 270 feet per second at 10000 ft. Note that for this flight condition, minimum-drag restoring drives the pitch thrust vectoring nozzle to its deflection limit (-14.2°).

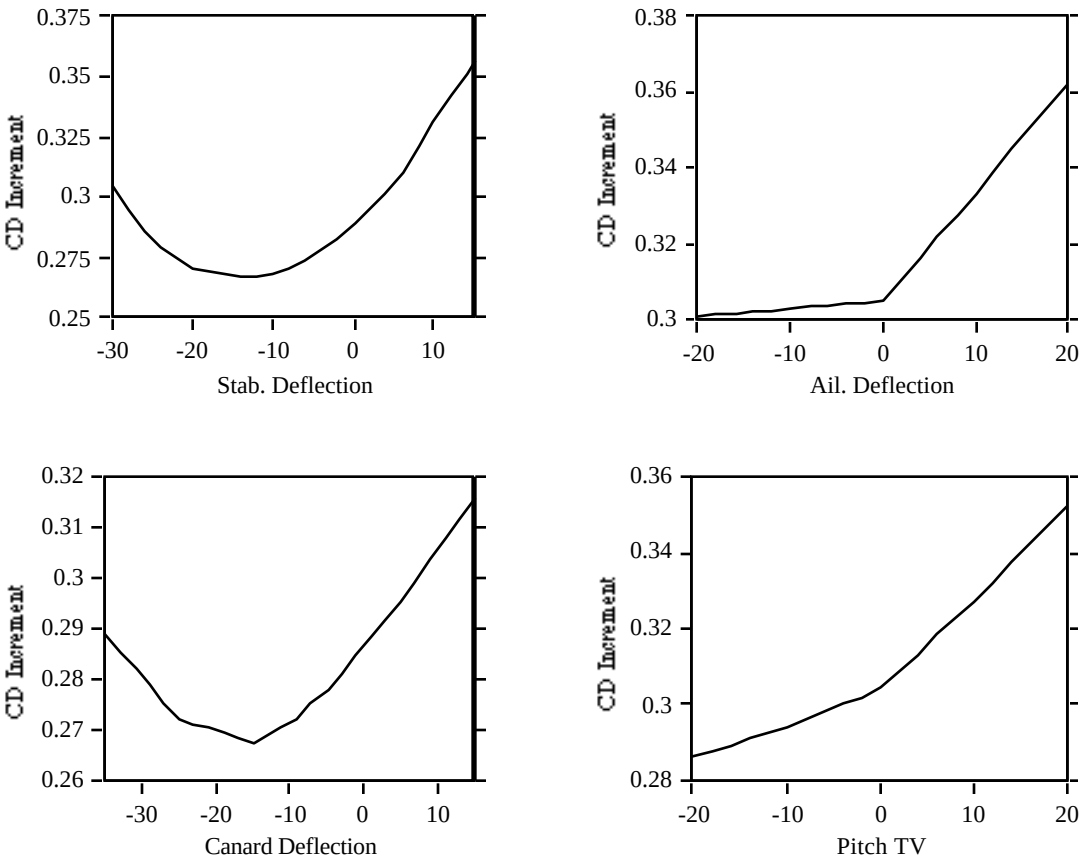

Figure 4.6 Drag Increments Due to Symmetric Control Deflections

These plots show the increments in drag due to symmetric stabilator, symmetric aileron, symmetric canard, and symmetric pitch thrust vectoring for a flight condition of 15 deg angle of attack, 0.25 Mach number, and 10000 ft altitude.

Thus, at the instant when the thrust vectoring nozzles reach their position limits, the restoring algorithm is still commanding negative changes in deflection in order to continue decreasing drag. However, because the shifted constraint in this direction is zero, the calculated scaling factor becomes zero, so that the remaining controls are prevented from being restored toward their minimum drag configurations. This example represents a condition in which the large magnitude deflections may be a high price to pay for trying to achieve a minimum-drag control configuration..

In summary, Chapter 3 introduced the idea of using a frame-wise direct control allocation scheme in order to include non-linear control effects and to compensate for their deflection rate capabilities as well as their position limits. For this reason, the algorithm was named Control Allocation with Rate Limiting (CARL). This chapter has built upon the CARL algorithm by presenting a restoring technique which takes advantage of the non-linear control effectiveness data to drive the controls toward some desired configuration, thus, alleviating the control wind-up problem which would otherwise occur. The building blocks for a basic CARL algorithm are now complete. The next two chapters will discuss further enhancements which are readily implemented. Chapter 5 will present the idea of control reconfiguration in response to a failure, and Chapter 6 will discuss the modifications necessary when control actuator dynamics are introduced.

CHAPTER 5.

Control Allocation with Adaptive Failure Control

In the age of modern aircraft and fly-by-wire control systems, the inclusion of mechanical backup systems for handling instances of control failures is becoming more uncommon. As a result, pilots rely on the failure immunities designed into these systems and be assured that, should such failures occur, the aircraft can maintain adequate flying qualities long enough for a safe ejection or an emergency landing. One of the more obvious advantages to control allocation schemes is the idea that once a control failure is detected, it can be dropped from the vector of controls to allocate, and the remaining controls can be reconfigured to obtain the desired moments. The commanded moments will still be attainable providing that the remaining control surfaces have enough control power, and that the remaining control effectiveness matrix defines a 3-Dimensional moment space (ie. the control effectiveness matrix has at least 3 linearly independent columns). The aircraft will then be able to recover from such failures with perhaps a minor degradation in performance and handling qualities. This chapter discusses the algorithms specific to Control Allocation with Rate Limiting required to provide a reconfigurable control allocation scheme.

5.1 Failure Immunity and Failure Safety Requirements

Flying qualities specifications give general guidelines for designing failure-immune and failure-safe flight control systems so that no subsystem failure results in a dangerous environment to the pilot. Reference 15 outlines these guidelines from MIL-F-9490D as follows:

“Failure immunity requires that no failure, not extremely remote, can result in any

of the following before a pilot or safety device can react:

1) Flutter, divergence, or other aeroelastic instabilities within the permissible flight envelope of the aircraft, or a structural damping coefficient for any critical flutter mode below the fail-safe stability limit of MIL-F-8870.

2) Uncontrollable motions of the aircraft within its permissible flight envelope, or maneuvers which generate limit airframe loads.

3) Inability to safely land the aircraft.

4) Any asymmetric, unsynchronized, unusual operation or lack of operation of flight controls that produces operation below FCS Operational State III.

5) Exceedence of the permissible flight envelope or inability to return to the service flight envelope."

The purpose of these guidelines is to ensure that any failures not considered extremely remote, do not result in an in-flight hazard for the pilot. In most cases, the flight control computer must constantly test for potential hazard-creating failures and react to them appropriately without any input from the pilot. Some extreme circumstances, however, may require that the pilot deactivate the failed controls or subsystem manually, or override them with the available cockpit inceptors.

In addition to the 5 previously mentioned specifications, the military guideline also requires that failures due to unforeseen natural occurrences like lightning strikes, or other induced environments such as enemy fire, do not result in flying qualities below level 3 (ie. the aircraft will still be able to be controlled safely, but it may require excessive workload by the pilot and may not be able to adequately perform its mission). In following these guidelines, many modern aircraft control systems are designed with redundant subsystems to include fail-operate, fail-safe operation. That is, after one type of failure (electric or hydraulic), the subsystem in question remains operative, and after a second similar failure, the subsystem reverts to a safe mode. In the case of an actuator for instance, this safe mode may require that the control surface be free to move as the hinge moments change, and is commonly referred to as a damper mode[15]. While this philosophy handles internal failures in the actuators, electrical systems, or hydraulic systems very well, it does not provide an adequate solution for cases in which a control surface is externally damaged due to an

enemy encounter or natural occurrence. These situations typically require the pilot to either disable the offending system, or in the case of physical damage due to enemy fire, to manually override the effects due to the failure.

For this reason, the idea of reconfigurable control allocation schemes has become an important design consideration. As an example, in the event of physical damage to an aileron (leaving it either inoperable in some non-zero position, or resulting in a complete loss of the control surface), the control law must be able to recognize the failure. It can then send this information to some control allocation scheme, which can then ignore the control in question and continue allocating the remaining controls. In allocation schemes which calculate the required controls based on moment commands, the remaining functional controls are allocated to satisfy the pilot commanded inputs, and the failed control(s) will produce an error between the obtained moments and commanded moments. This error would then appear to the control law as a disturbance and would have to be compensated for accordingly.

The Control Allocation with Rate Limiting (CARL) algorithm allocates changes in control deflections based on some desired change in moments. The desired change in moment vector is found according to the error between the obtained moments in the previous sample frame and the commanded moments in the current sample frame. As a result, a self-correcting algorithm is achieved in which the effects due to the failed controls are canceled. This fact will be demonstrated in the next section with a simple, hypothetical example.

5.2 Control Reconfiguration for the CARL Algorithms

The algorithms used to handle control allocation with failed control surfaces are identical to those developed in Chapters 3 and 4 with some additional “book keeping” logic to keep track of the functional and non-functional controls. The responsibility of control allocation is to utilize this information, which is assumed to be given by the control law, to adjust the size of the control effectiveness matrix, the number of controls, and their minimum and maximum constraints accordingly.

The key concept which allows CARL to adapt to failures is the fact that it allocates changes in controls based on some desired change in moments. That is, given a local control effectiveness matrix $\mathbf{B}_{3xm}$, as a function of aircraft states and control deflections, and a desired change in moment vector $\mathbf{m}$, a m -Dimensional control vector $\mathbf{u}$ can be found such that

$$\mathbf{m} = \mathbf{B}\ \mathbf{u} \tag{5.1}$$

subject to the controls' minimum and maximum constraints. The $\mathbf{m}$ input to the control allocation software is based on the difference in the current moment commands produced by the control law $\mathbf{m}_{\mathbf{c}k}$, and the allocated moments from the previous sample frame $\mathbf{m}_{\mathbf{a}k-1}$. The moment vector $\mathbf{m}_{\mathbf{a}k-1}$, is generally unknown and must be approximated using either global effectiveness data referenced from the controls' origins or secant-slope data as discussed in Chapter 2, and the allocated control positions, $\mathbf{u}_{\mathbf{a}k-1}$, such that,

$$\mathbf{m}_{\mathbf{a}k-1} = \mathbf{B}\mathbf{u}_{\mathbf{a}k-1} \tag{5.2}$$

In situations where a control surface may be damaged to the point where its position cannot be measured, the resulting aircraft accelerations could be sensed and converted into attained moments.

Under ideal circumstances, where the control effectiveness data is assumed to be exact, the allocated controls produce the commanded moments exactly, and the $\mathbf{m}$ vector is strictly a result of some dynamic process. A failed control surface (either a hardover, jammed actuator or physical damage caused by enemy encounters), would then show up as an error between the commanded moments and attained moments.

As a simple example, consider an aircraft in a trimmed flight condition, and assume a control failure is detected in the k-1 frame. This information is passed on to the control allocation software, and the controls are reconfigured as necessary to achieve the commanded moments. Divide the total moment vector for this frame, $\mathbf{m}_{k-1,}$ into two parts; one being the part that is acquired with control allocation, $\mathbf{m}_{\mathbf{a}k-1}$, and the other being the unknown moment vector induced by the failed control(s), $\mathbf{m}_{\mathbf{f}k-1}$. As time proceeds on to the kth frame, the commanded moment vector remains constant (because of the static flight

condition). Furthermore, because the control effectiveness data is assumed to be exact, $\mathbf{m_a}_{k-1} = \mathbf{m_c}_{k-1} = \mathbf{m_c}_{k}$. The control allocation algorithms find a desired change in moment for the current frame k, according to:

$$\begin{aligned} \mathbf{m}_k &= \mathbf{m_c}_k - \mathbf{m}_{k-1} && (5.3a) \\ &= \mathbf{m_c}_k - (\mathbf{m_a}_{k-1} + \mathbf{m_f}_{k-1}) && (5.3b) \\ &= -(\mathbf{m_f}_{k-1}) && (5.3c) \end{aligned}$$

Thus, even in the trimmed case, there would exist a non-zero $\mathbf{m}_k$, which the allocator would use to offset the effects of the failed controls and drive the error to zero.

In general, the aircraft may be involved in some maneuver, and the control effectiveness data will not be known exactly. Yet, this control allocation algorithm will still cancel the effects of any failed controls. In this case however, the desired change in moment vector contains the terms needed to cancel the errors due to the failed control(s), the error associated with the control effectiveness data, and the necessary changes in moments required to perform the desired maneuver.

5.3 Control Allocation with Adaptive Failure Control: An Example

This section will demonstrate the ability of Control Allocation with Rate Limiting to reconfigure the controls in response to a failure. In this example, the previous time-asymmetric maneuver depicted in the second plot of Figure 3.1 is used to provide the inputs that would normally be calculated by some control law with adaptive failure logic. The flight conditions for this maneuver are identical to those used in Chapters 3 and 4 (400 ft/sec, 8 deg angle of attack, 10000 ft altitude). At t = 3 sec, a trailing edge up hardover is simulated in the left aileron. The failure indicator for this particular control surface is assumed to be given by the control law and is passed to the CARL algorithms for reconfiguration.

Figure 5.1 shows the control time histories for this simulation using the minimum-norm restoring technique described in Chapter 4.

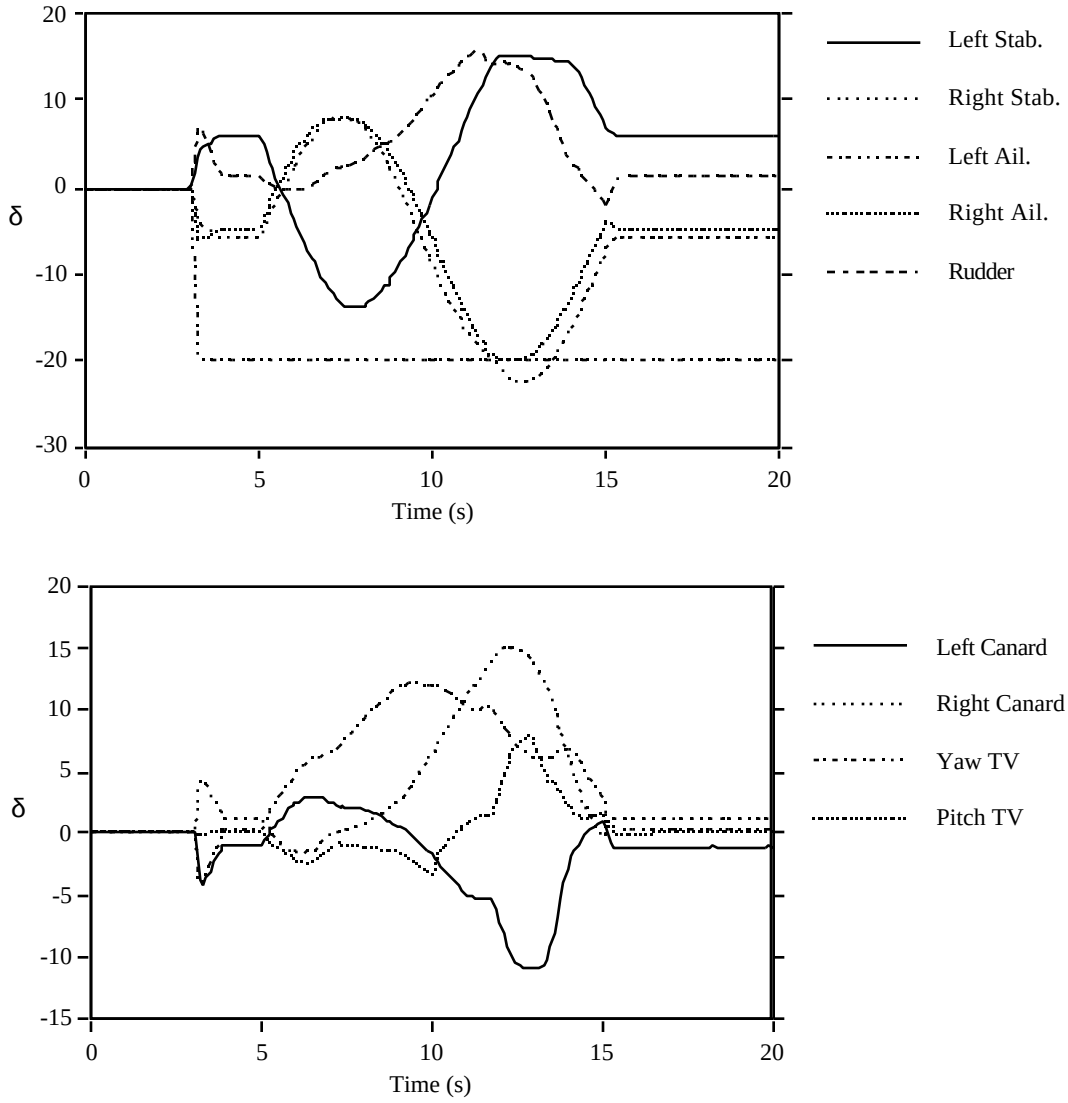

Figure 5.1 Control Time Histories with a Control Failure
These plots demonstrate the control reconfiguration capabilities of Control Allocation with Rate Limiting in response to a trailing edge up hardover in the left aileron. The moment commands followed were those associated with the time-asymmetric maneuver introduced in Chapter 3.

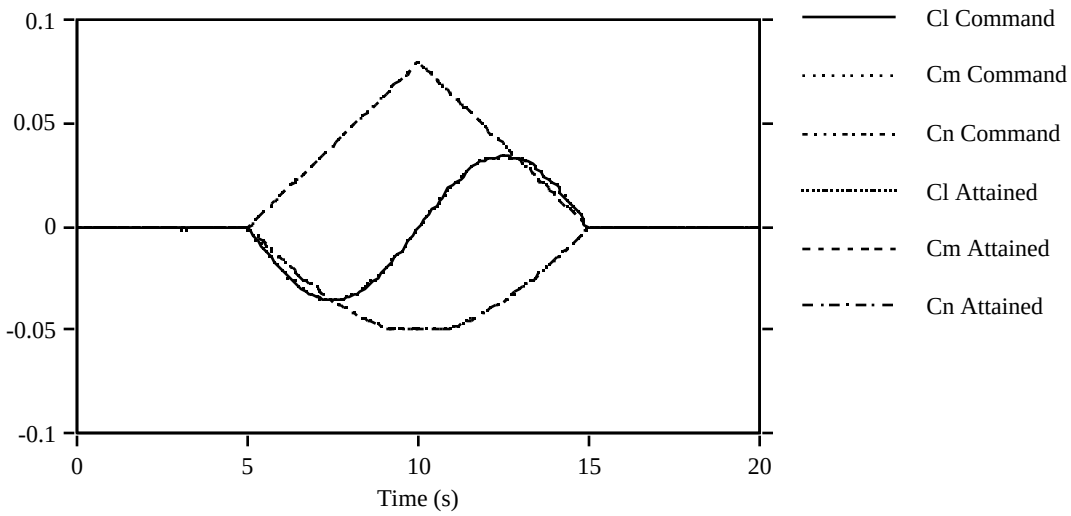

Figure 5.2 Moment Time Histories with a Control Failure
This plot shows that with the control failure depicted in Figure 5.1, Control Allocation with Rate Limiting is still able to follow the commanded moments very accurately. The error between the commanded and attained moments at the instant of the failure are insignificant when compared to the magnitudes of the commanded moments and are therefore difficult to analyze in this figure. A more detailed error analysis is presented in Chapter 6.

In contrast to the results presented in Figure 4.3 however, the remaining controls are required to have significantly larger magnitude deflections, (some of the controls actually become position saturated). This result is expected since the left aileron is constrained to remain at its negative position limit. In addition, the failure requires that the controls be used differentially after the maneuver to cancel the rolling and yawing moment effects of the left aileron.

Figure 5.2 compares the commanded moments with the attained moments. At the instant of control failure, there is a small error spike in all three moment axes. The error is due to the fact that the control allocation routines only receive the position of the control surface at the beginning of each frame and have no way of extrapolating its location to the end of the frame. However, once the left aileron reaches its physical position limits, the errors are canceled by the “delta” moment allocation scheme. It is important to emphasize that this simulation was performed without a control law, and the error dynamics are therefore a result of the discrete allocation process only. Implementation of a control law may provide some additional compensation to further improve the error dynamics.

Another interesting aspect of Control Allocation with Rate Limiting is revealed in Figure 5.1. Because the left aileron is rate saturated, intuition would suggest that the right aileron should be rate saturated as well in an effort to cancel the effects due to the left aileron. However, the right aileron is allocated at a noticeably slower rate. One of the reasons is the fact that all of the remaining controls are collectively utilized to cancel the effects due to the failure so that the right aileron is generally not required to travel at its maximum rate. This result reemphasizes the fact that Control Allocation with Rate Limiting utilizes the most efficient capabilities of the controls in generating moment rates. Another contributing factor arises from the effects that actuator dynamics have on the control allocation system. These effects will be presented in more detail in Chapter 6.

CHAPTER 6.

Control Allocation and Actuator Dynamics

Until now, a major assumption in developing the Control Allocation with Rate Limiting schemes was that the control deflection commands, which were calculated by adding the allocated changes in controls to the previous frame's control positions, could always be obtained. This assumption was used because the allocated changes in controls were guaranteed not to violate any rate or position limit constraints. The position limits were taken as the absolute minimum and maximum control deflections allowed for any particular flight condition, and could be imposed by either the controls' physical limits or software limits specific to some flight condition. Likewise, the rate limits were imposed by using the maximum rate for both positive and negative deflection directions. Thus, if the rate limit capabilities resulted in the most restrictive constraint, then the largest magnitude command that control allocation could produce would be obtained by deflecting the control at its maximum rate for the entire sample period. What was not modeled in these algorithms however, was the fact that the control surfaces are manipulated by either hydraulic or electric actuators, and constitute a dynamic system which cannot produce the infinite accelerations that were assumed with control allocation. In other words, if a control was initially at rest, and later commanded to move at its maximum rate in some direction for a specified amount of time t, it would gradually build up speed until it reached the commanded rate. The final position of the control would therefore not be the same as that calculated using the commanded rate and the time during which it was instructed to move. This chapter will introduce the effects of actuator dynamics on Control Allocation with Rate Limiting and present the necessary modifications required to compensate for these dynamics.

6.1 The Actuator Model

In most modern aircraft, the need for mechanical linkages to transfer pilot commands into control surface deflections has been eliminated. Some of the more obvious reasons for discarding the old mechanical control systems involve the weight savings and the simple fact that most tactical aircraft are designed to be either unstable or nearly unstable, requiring a digital computer for stability and control augmentation. In addition, when considering that such aircraft can have as many as 20 aircraft controls for 3 primary cockpit inceptors, a mechanical linkage system would be very difficult to design and very impractical in terms of weight, cost, and functionality. As a result, modern aircraft control surfaces are typically driven by either hydraulic, electric, electro-hydraulic, or any other variance of the so-called actuator.

The dynamics of one of the more "simple" hydraulic actuator models described in reference 16 are actually quite complex. This particular example models the spool valve and hydraulic ram components of the actuator based on flow rate and hydraulic pressure relations. In addition, the control surface and actuator piston are treated collectively as a spring, mass, and damper system. Thus, even this simple model describing the actuator/control surface system may contain as many as 4 dynamic states. For practical purposes however, these dynamics are often simplified to produce a simple first-order lag filter[16].

The current implementation of the CARL algorithms assume a first-order lag actuator model with non-linear elements to account for actuator position and rate saturation. The differential equation for this model is given by:

$$\dot{u} = \frac{1}{\tau}\left(u_c - u\right) \tag{6.1}$$

where u represents the instantaneous control position, u_c represents the commanded position, and τ represents the actuator time constant. The block diagram model with saturation elements is shown in Figure 6.1.

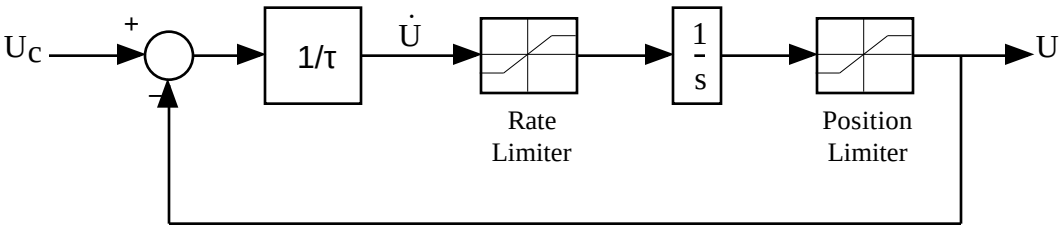

Figure 6.1 A First Order Actuator Model

This block diagram represents a first order mathematical model for an actuator with non-linear elements for rate limiting and position limiting.

6.2 The Actuator Response

One of the advantages of using the simple dynamic model given by Eq. 6.1 and depicted in Figure 6.1 is that an analytical expression for the position with respect to time can readily be found. By assuming zero initial conditions and a commanded step input of magnitude u_c, the actuator response follows:

$$u(t) = u_c\,(1 - \mathrm{e}^{-t/\tau}) \tag{6.2}$$

provided that the actuator does not become rate or position-saturated. Eq. 6.2 can then be differentiated with respect to time to find the actuator velocity as a function of time.

$$\dot{u}(t) = \frac{u_c\,\mathrm{e}^{-t/\tau}}{\tau} \tag{6.3}$$

which is the same result obtained had Eq. 6.2 been substituted directly into Eq. 6.1.

Now that the time response of the actuator is known, a few important observations should be made. First, Eq. 6.1 reveals that the actuator rate at each instant in time is proportional to the difference between the commanded position and the actual position. Since the initial position is assumed to be zero here, the initial difference is in fact the magnitude of the command. After this instant, as the position approaches the commanded signal, the instantaneous rates approach zero. These trends are easily verified through equations 6.2 and 6.3. A generic response for an actuator of this type is shown in Figure 6.2 to graphically depict these trends. Note that had the actuator velocity remained constant throughout time (as was previously assumed in control allocation), then it would have traveled along the dashed line representing the initial slope (given as u_c/τ), and would have reached the steady-state commanded position sooner.

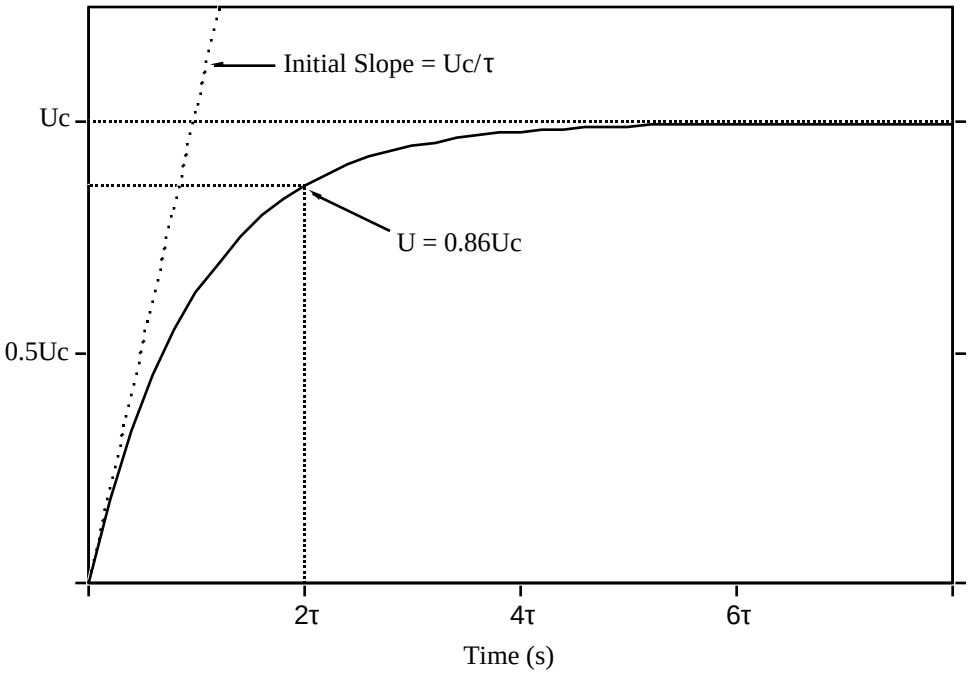

Figure 6.2 Time Response of a First-Order Actuator
This plot shows the general shape of the position response for the actuator model shown in Figure 6.1 with no position or rate saturation.

6.3 The Actuator Response For Discrete Signals

The dynamic equations for a first-order actuator model have been presented and the response to a commanded step $u_{c,}$ has been analyzed for the continuous-time system. Control laws on modern aircraft however, are implemented in digital computers which sample data and send control signals to the actuators at discrete intervals of time. One unique feature of these so-called sampled data systems is that between samples, the continuous plant (or aircraft) runs in an open-loop environment whose inputs are supplied by a digital to analog (D/A) device. The purpose of these devices is to take the last discrete input calculated by the flight control computer and perform some type of extrapolation so as to provide a continuous signal to the aircraft between samples. One of the most common types of D/A devices is the zero-order hold, for which the discrete inputs are held constant over the sampling interval. Thus, the continuous system sees a series of finite step inputs.

Recall that the Control Allocation with Rate Limiting algorithm commands control positions based on their previous positions and some allocated change in the control vector, which may or may not include a restoring term to drive the controls toward a desired configuration. For this discussion, the allocated control vector and the restoring vector will be convolved into one variable, signified by $\mathbf{U}$ such that:

$$\mathbf{u}_k = \mathbf{u}_{k-1} + \mathbf{U} \tag{6.4}$$

Since these algorithms will also be implemented in the flight control computer, they will operate in a discrete-time domain as well. This type of sampled data scenario for one control actuator is depicted in Figure 6.3. Note that at each sample instant (assuming no transport delay between the time data is sampled and the time that input signals are sent to the aircraft), the commanded input is found by adding the allocated “delta” control to the current position and sent to the hold device. Inside the continuous model of the actuator, there is a negative feedback of the current position that produces the error signal between the commanded and actual positions. This negative feedback of the position will cancel the u_{k-1} term in Eq. 6.4 at the sampling instant so that the actuator essentially sees a step input with magnitude U at the beginning of each sample interval.

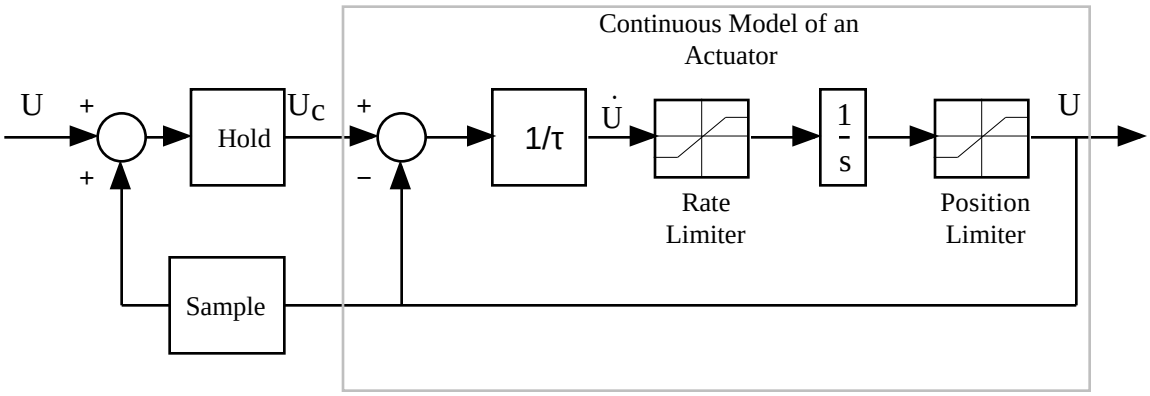

Figure 6.3 A Sampled Data System

This block diagram represents the actuator model from Figure 6.1 when connected to a digital allocation scheme such as Control Allocation with Rate Limiting. The components within the dashed lines represent the continuous-time system.

From Figure 6.2, the amount of actuator travel for a step of magnitude u_c, is expressed in terms of the number of time constants passed. That is, after a period of time equal to τ, the actuator has moved to 63.2% of its commanded position, and at 2τ, it has moved to 86.5% of its commanded value. While the response, in general, will never reach the commanded input, it does approach this so-called steady-state value as time approaches infinity, so that for practical purposes, there exist some time t, at which the error signal is negligible. It can be concluded however, that if the sample period is on the same order of magnitude as the actuator time constant, then there will be a significant error between the position obtained at the end of the sample frame and the initial commanded position. Figure 6.4 depicts this graphically for a situation in which the sample period is the same as the actuator time constant. At $t = 0$, the actuator receives a step command which corresponds to some desired rate multiplied by the sampling interval. The initial rate of the actuator given by Eq. 6.3 is u_c/τ, and decreases as the error between the step command and the actual position decreases. The final position obtained at the end of the sample period is approximately 63.2% of the commanded signal, so that the average rate obtained is significantly less than the commanded rate. At the beginning of the next frame, the next commanded step is calculated by adding the allocated U to the actuator position at that time, and the actuator responds in the same fashion as it did in the previous frame.

Note that without the actuator dynamics, the effective rate of the actuator would have been much faster. This fact presents a serious problem since the control allocation algorithms are calculating changes in controls based on maximum rate capabilities that must be obtained during the sample interval in order to achieve the desired moment rates. What is required then is a method to overdrive the control actuators using some vector of gains so that they obtain the commanded positions by the end of the sample interval. The procedure for finding the required gain for one control is outlined as follows:

1) Evaluate Eq. 6.2 over the sampling interval T, with the U_c input calculated by Control Allocation with Rate Limiting:

$$U(T) = U_c\left(1 - e^{-T/\tau}\right) \tag{6.5}$$

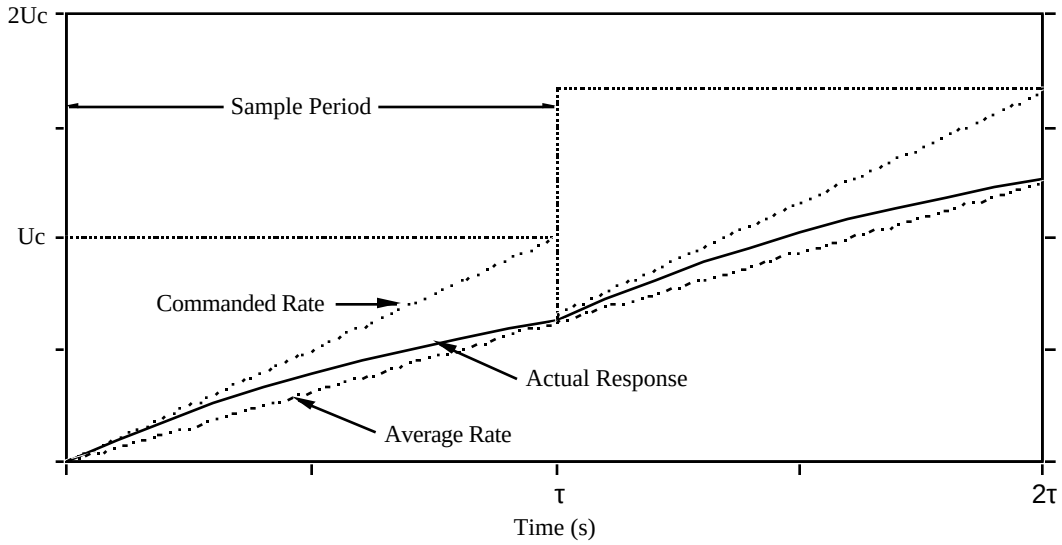

Figure 6.4 Response of a Sampled Data System

This figure shows the response of the actuator described in Figure 6.3 for two sample periods. Note that the sample period in this plot is equal to the actuator time constant. As a result, the actual response does not reach its steady state value by the beginning of the next sample frame, and the average rate is not equal to the commanded rate.

Note that when the sample period is close to the same order of magnitude as the time constant, then $U(T)$ will be less than U_c.

2) Now, it is desired to amplify the commanded input U_c,with some gain K, such that the obtained output $U(T)$, is equal to the commanded input:

$$U(T) = U_c = K\ U_c\left(1 - e^{-T/\tau}\right) \tag{6.6}$$

3) The unknown gain is found to be:

$$K = \frac{1}{\left(1 - e^{-T/\tau}\right)} \tag{6.7}$$

Thus, by applying the gain K given by Eq. 6.7 to each allocated U_c, it is ensured that the amount of deflection obtained by each actuator is the same as that commanded by the control allocation software.

Of course, this method assumes that the actuator does not become rate saturated during the sample period. If the compensated commanded input attempts to exceed the actuator's rate capabilities, then rate limiting will occur and the obtained deflection will be somewhat less than the commanded deflection. The point at which rate limiting results from the compensated command is dependent on the amount of rate saturation commanded by the control allocation algorithms as well as the ratio of the sample period to actuator time constant and is plotted in Figure 6.5. To the right of this curve, rate limiting occurs and the commanded deflection is not obtained. To the left, no rate limiting occurs and the commanded deflection is obtained exactly. Note that at 100% commanded rate, the T/τ ratio approaches zero so that the gain as calculated in Eq. 6.7 approaches infinity. Thus, in mathematical terms, Eq. 6.6 cannot be solved. For all practical purposes however, a large enough gain will cause the actuator to be rate limited across the entire sample period, so that the commanded deflections will be obtained. The resulting block diagram for the control allocation scheme implemented here is shown in Figure 6.6.

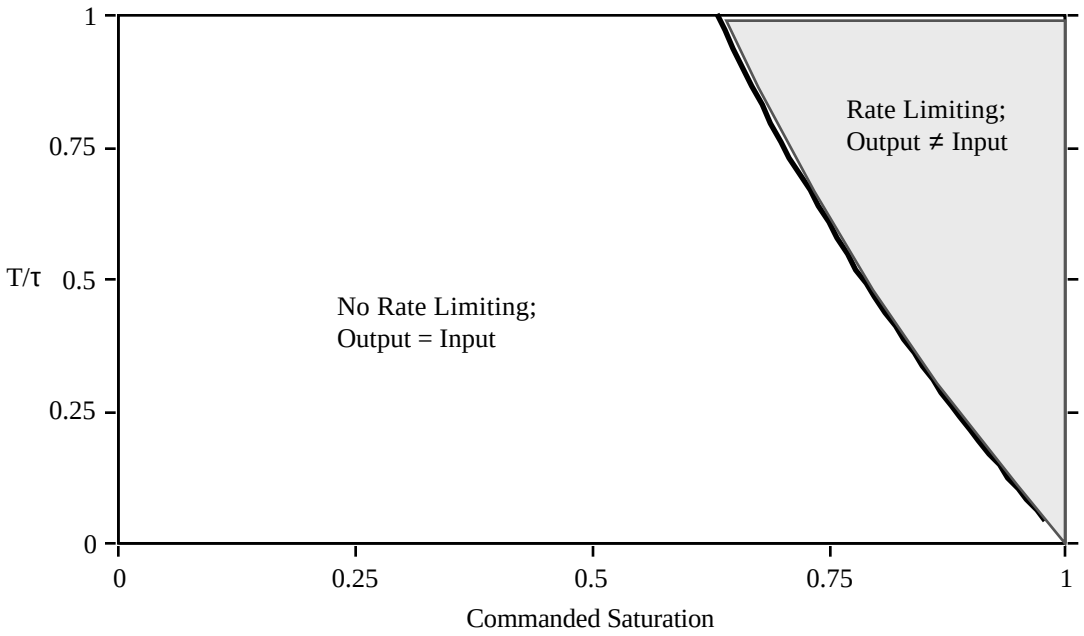

Figure 6.5 Regions Where Actuator Rate Limiting Occurs
This plot shows the regions where actuator rate limiting occurs as a function of the amount of commanded rate saturation and the sample period to time constant ratio. This information was obtained using the actuator compensation gain calculated in Eq. 6.7.

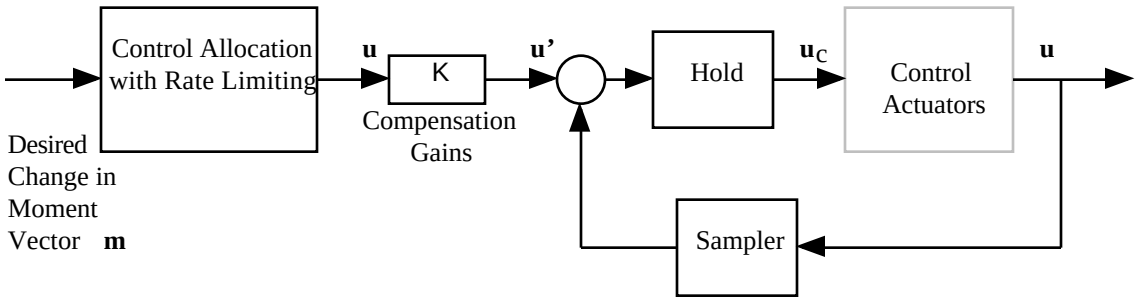

Figure 6.6 Control Allocation with Rate Limiting and Actuator Dynamics

This block diagram represents the way that Control Allocation with Rate Limiting is implemented when actuator dynamics are modeled. The compensation gain vector represented by $\mathbf{K}$ ensures that the commanded deflection is obtained so long as no actuator rate limiting occurs.

6.4 Simulation of a Control Failure with Actuator Dynamics

The purpose of this section is to demonstrate the effects that actuator dynamics have on the control allocation/control actuator system described by Figure 6.6. The example presented is the same as that used in Chapter 5 to demonstrate the control reconfiguration capabilities of Control Allocation with Rate Limiting. All actuators have a T/τ ratio of 0.2525. Thus, after the compensation gains are applied, the onset of rate limiting will occur when the allocated controls are commanded at approximately 90% of their maximum rates (see Figure 6.5). Since the full time histories for this example can be referenced in Figures 5.1 and 5.2, only the relevant time range around the left aileron hard-over will be shown here. Figure 6.7 compares the responses of the right aileron, with and without the gain compensation for the actuator dynamics, to the response of the left aileron, which was given a trailing edge up hardover command at $t = 3$ sec.

The saturation level in the discrete "delta" moment space, (ie. how much of the moment rate-generating capabilities are utilized), for the compensated and uncompensated systems is shown in Figure 6.8. Because the compensation gains have effectively increased the control deflection rates available to Control Allocation with Rate Limiting, the required moment saturation is considerably less for the compensated system.

In Figure 6.9, the control-generated rolling moment coefficients are compared for the compensated and uncompensated systems. It shows that the increase in rate capabilities achieved from this type of overdrive compensation has helped significantly in decreasing the error between the commanded and attained moment during the aileron hard-over. This chapter concludes the discussion of the theory and mathematical algorithms associated with Control Allocation with Rate Limiting. The following chapter will describe the Control Allocation software from a functional perspective.

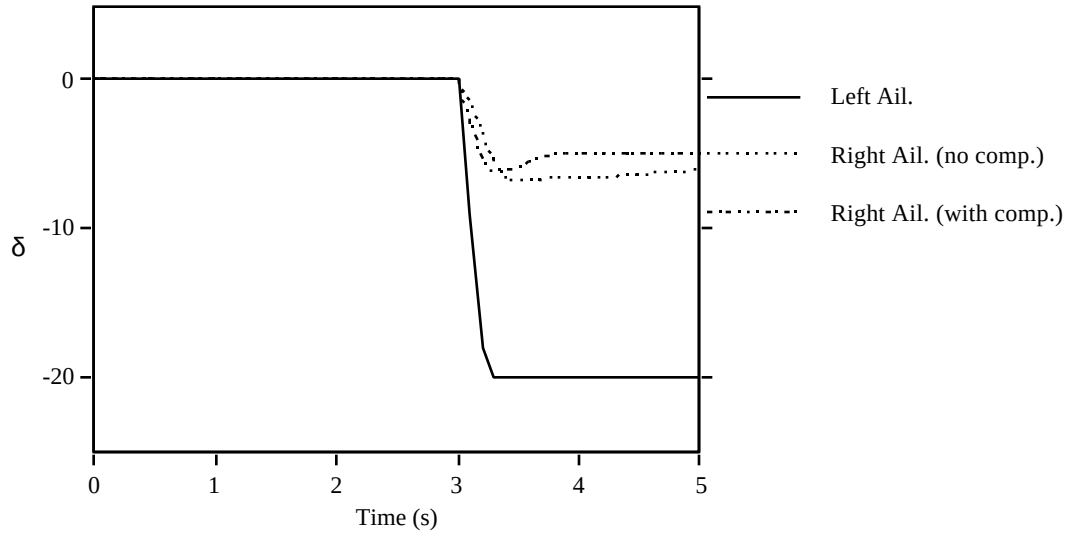

Figure 6.7 Simulation of a Control Failure with Actuator Dynamics (Control Effects)
This plot shows the effect that the compensation gains have on the control allocation/control actuator system shown in Figure 6.6.

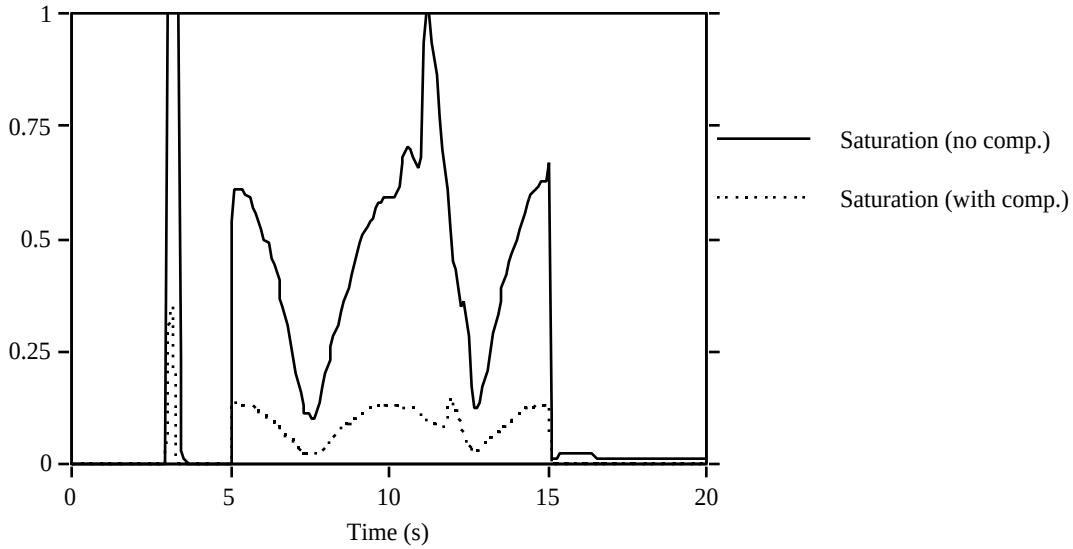

Figure 6.8 Percent Moment Saturation For an Aileron Hard-Over Simulation
This plot compares the amount of moment saturation (ie. how close the desired moment vector is to the AMS boundary), for the Control Allocation/Control Actuator system shown in Figure 6.6 with and without the compensation gains implemented. The effect of the gain vector is to increase the control deflection rates available to control allocation, so that the saturation level in moment space decreases.

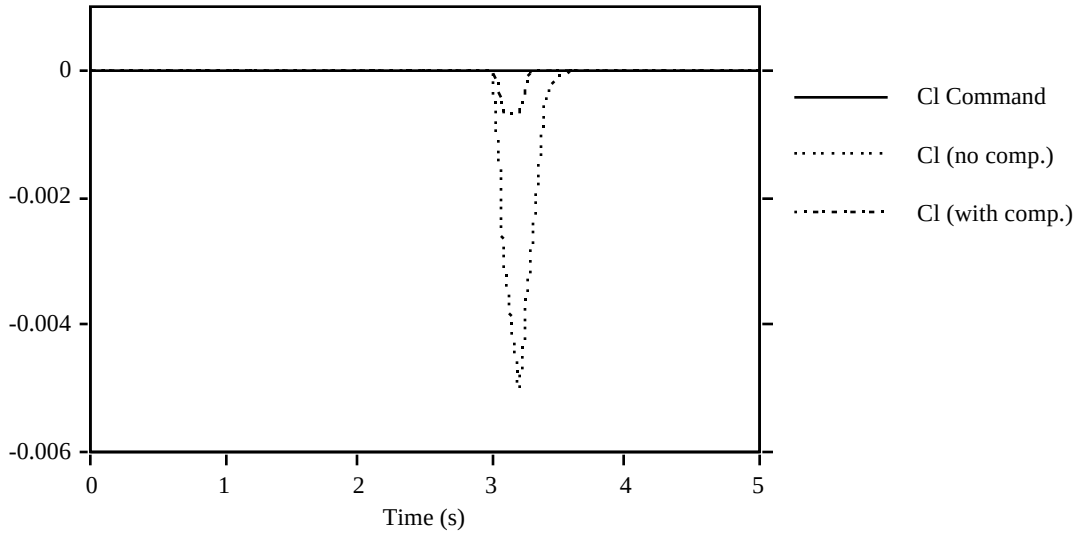

Figure 6.9 Simulation of a Control Failure with Actuator Dynamics (Moment Effects)
This plot shows the effect that the overdrive gain has on the control allocation/control actuator system shown in Figure 6.6. The extra rate capabilities achieved with the actuator dynamics compensation helps considerably in decreasing the error in rolling moment caused by the left aileron hard-over.

CHAPTER 7.

The Control Allocation with Rate Limiting Software

One of the goals of this research was to generalize the control allocation algorithms and to compile them into an independent software suite (written in ansi-standard FORTRAN) so that future implementations of these techniques for different aircraft would require minimal time and effort. The code is therefore divided into two segments. The first segment consists of all of the generic routines. They perform the general constrained control allocation procedures which apply to any aircraft, and are as follows:

1.) CONALLO	The main control allocation executive.
2.) ALLODIAGSOUT	An optional module that writes diagnostic information to a file.
3.) RESTORE_U	Performs various restoring techniques as described in Chapter 4.
4.) GET_FACET	Determines the facet geometry of the AMS given a pair of face-defining controls.
5.) GET_MAT	Stores the facet geometry for a given pair of controls in a matrix.
6.) GET_U	Finds the allocated control vector given the appropriate facet geometry, control constraints, and desired moments.
7.) PINVB4	Calculates the Right Pseudo-inverse of a 4 by *m* rank-4 matrix.
8.) INVMAT3	Calculates the inverse of a 3 by 3 invertible matrix.
9.) INVMAT4	Calculates the inverse of a 4 by 4 invertible matrix.
10.) D3	Calculates the determinant of a 3 by 3 matrix.

These routines will typically remain unchanged from one implementation to another. The

constrained control allocation techniques however, also require knowledge of control effectiveness data and control constraints, (which are aircraft dependent). Therefore, a second segment of user-defined routines is required, and consists of:

1.) A_C$GETUEFF — Returns the effectiveness of a given control for a given moment (objective) axis.

2.) A_C$GETCSTR — Calculates the vectors of minimum and maximum position limits and rate limits.

The developer is free to design these routines to his or her liking, provided that they adhere to the calling conventions used in the CONALLO executive. In addition, a set of initialization routines may be required to set up the control effectiveness lookup tables and other aircraft dependent data. These routines are not called directly by CONALLO and are therefore omitted from this discussion. While the calling conventions and detailed functional descriptions for the CARL software, along with the code listings, are documented in Appendix II, a basic overview of the software architecture and features is presented in this chapter.

7.1 Generic Routines

The 10 generic CARL routines previously listed have been designed in such a way to allow easy implementation of different aircraft data without the need to make significant changes to the underlying control allocation code. This feature is made possible by the fact that the flight control computer (or flight simulator) must maintain a vector of allocatable controls (**U**) to be accessed by CARL. Once this vector is passed to CARL, its physical meaning is lost. That is, it is no longer important that the first entry of the control vector is an aileron for instance, but only that it has a certain amount of effectiveness in generating the 3 aircraft moments and any additional objective (if restoring is used). The responsibility of maintaining information relating entries in the **U** vector to actual control surfaces on the aircraft is left for the control laws to handle. In addition to the generalized nature that this feature provides, it also allows great freedom in deciding which controls to allocate. As an example, the aircraft flaps are generally regarded as secondary controls and are not used for primary maneuvering capabilities. They can therefore be omitted from the **U** vector, and alternatively be controlled by some other type of control mixing logic.

The CARL main executive (CONALLO) serves primarily as a front end between the flight control computer and the actual control allocation algorithms. It currently supports as many as 20 aircraft controls and can allocate either changes in controls using non-linear effectiveness data and desired changes in moments (Control Allocation with Rate Limiting), or can allocate absolute control surface commands based on global (slope at the origin) data and absolute moment commands. It also has the ability to reconfigure the allocatable controls in response to a reported failure and can invoke control restoring whenever there are sufficient rate capabilities remaining after the desired moments have been accounted for.

The ability to switch between Direct Control Allocation and Control Allocation with Rate Limiting is handled by the USE_GLOBALS flag. When its value is true, global control effectiveness data is used and direct allocation is performed. This algorithm is similar to that described in Reference 6 except that it is more robust in terms of error handling and rare (but possible) exceptions to the theory. For instance, the modern algorithms can handle situations where the origin and desired moment lie on the same bounding facet. This case is a rare circumstance which can only occur when one or more controls are position saturated, yet, it presented a serious problem in earlier algorithms. If the USE_GLOBALS flag is false, then the discrete rate limiting allocation scheme described in Chapter 3 is invoked. During this mode, additional features such as control reconfiguration and control restoring can be enabled.

Control reconfiguration is accomplished by using a vector of integer flags (**IFAIL**) whose entries correspond to the controls in the **U** vector. As long as all of these flags are zero, (indicating that all of the control surfaces are functional), control allocation proceeds in a normal fashion. The existence of any non-zero element indicates that a particular control has failed. Regardless of the type of failure, CONALLO drops the offending control(s) from the **U** vector, and then allocates the remaining functional controls. Finally, if restoring is desired, and if there are sufficient rate capabilities remaining after the moment demands have been satisfied, CONALLO calls the RESTORE_U routine to apply minimum-objective restoring. The resulting allocated “delta” control deflection commands are then multiplied by their respective actuator compensation gains discussed in Chapter 6.

The direct control allocation concept is implemented within the GET_FACET and

GET_U routines. These modules comprise modifications made over several years of testing and debugging in an attempt to produce the most robust and efficient algorithms possible. Enhancements have been made to the method of determining facet geometries, checking facets for control allocation, and error reporting.

In Reference 3, a method is presented to determine, for a given pair of face-defining controls, the positions of the other controls required to generate the bounding facets of the AMS. This determination is made by finding a rotational transformation such that when applied to the original control effectiveness matrix, the faces defined by the pair of controls on the AMS lie perpendicular to the first axis defined by the first row of the transformed effectiveness matrix. The idea is that these two controls will then have no contribution along the rotated axis. Finding the deflections of the other controls needed to generate maximum and minimum moments in that direction simply requires an inspection of the signs of the transformed entries. That is, to maximize the moment contributions in the given direction, any controls having an effectiveness along that direction greater than zero would need to be at their maximum deflections. Likewise, any controls having negative effectiveness along the specified direction would need to be at their minimum deflections. Furthermore, since the only information required is that describing the controls' effects along the first transformed axis, only the first row of the unknown transformation matrix is required, resulting in a system of 2 equations with 3 unknowns. In the algorithm presented in Reference 3, one of the entries is arbitrarily set to 1 and the remaining two entries are found by solving a set of two linear equations for two unknowns. Unfortunately, the required 2 by 2 matrix inversion is often numerically ill-conditioned, and results in significant round-off errors within the hosting computer.

In the most recent version of GET_FACET, a more suitable algorithm is employed. Since the face defined by the two specified controls is essentially a linear combination of the two respective columns of the control effectiveness matrix, a normal vector to this face can be found by simply calculating the cross product of the two columns. The normal vector can then be scaled such that its magnitude is 1, resulting in a vector of direction cosines for a transformed axis, (perpendicular to the face), with respect to the original 3 moment axes. This vector then represents the first row of the transformation matrix described in

Reference 3, and upon multiplying it by the control effectiveness matrix, the signs of each entry in the resulting row can be inspected as before to find the required positions of the remaining control surfaces.

In some special cases, after multiplying the transformation by the effectiveness matrix, additional entries (besides those associated with the two specified controls) become zero. In other words, one or more of the controls are redundant, meaning that their effects can be duplicated by one or more other controls. In this case, the required positions of such controls are not easily determined, and each possible combination of their minimum and maximum deflections must be checked. A rather efficient algorithm for handling these so-called "special" controls is included in GET_FACET. The algorithm allows for any number of redundant controls. However, the number of possible minimum and maximum combinations to check increases according to 2^n, where n is the number of special controls. As an example, if there are 2 special controls, then there would be 4 possible combinations to account for when checking facets. Based on the binary facet coordinate nomenclature from Reference 1, the four combinations for the two redundant controls that must be checked would be {(0,0), (0,1), (1,0), (1,1)}. In the current implementation, a limit of 4 special controls is imposed on GET_FACET.

Once the facet geometry is calculated and stored as a matrix by GET_MAT, GET_U uses the desired moment vector (referenced from the origin in moment space), the facet geometry, and the vector of control constraints to check if the moments intersect the facet. This search algorithm, as described in Appendix II, fails when the origin lies on the facet because the matrix containing the facet geometry becomes singular. Therefore, a method of incorporating this singular case is also included. The remaining routines, (INVMAT3, INVMAT4, PINVB4, and D3) perform the generic matrix functions required by the CARL and restoring algorithms, and are described in detail in Appendix II as well. For quick reference, a very basic flowchart of the CARL software dictating the order in which each module is called is shown in Figure 7.1.

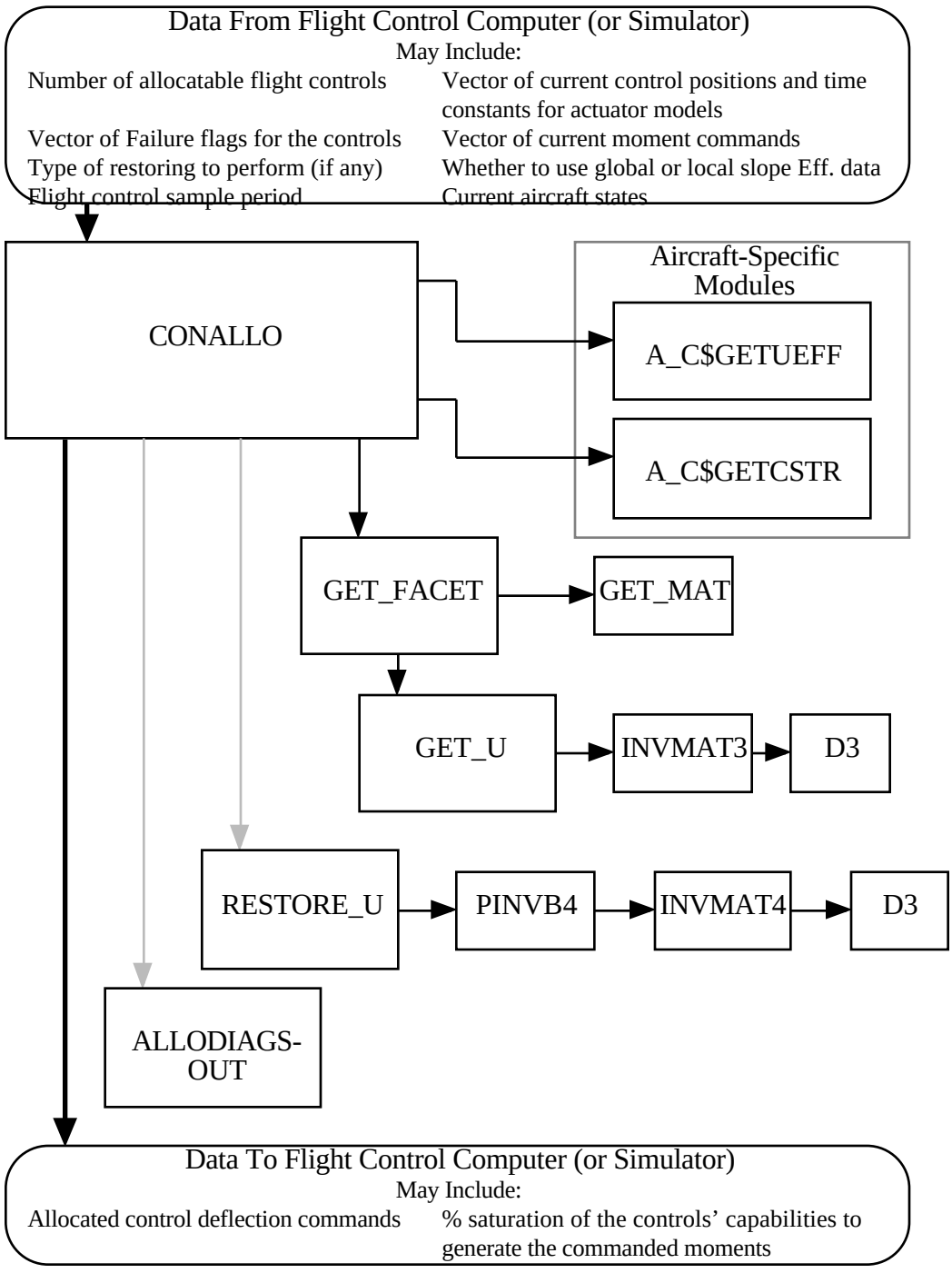

Figure 7.1 Architectural Diagram of the CARL Software
The paths colored in gray are optional and do not have to be executed every frame.

7.2 Aircraft-Specific Routines

The open framework design of the two aircraft-specific routines, A_C$GETUEFF, and A_C$GETCSTR, has been developed for two reasons. First, this type of design allows for easier integration of different aircraft models without having to modify the generic modules mentioned above. At present, two models have been successfully tested using this approach. One model, (an F-18 HARV), consists of a 5 control configuration, while the F-15 ACTIVE model, (which will be described in detail in Chapter 8), employs 9 controls. The second advantage to this type of architecture, is that it allows great flexibility in the methods used to calculate the aircraft-specific data as well as in the precision to which the data is calculated. For example, if the hinge moments are known for the aircraft control surfaces, A_C$GETCSTR could be built to return control rate limits as a function of the control hinge moment data, resulting in a more accurate description of control constraints. In addition, the control effectiveness lookups in A_C$GETUEFF could also account for any number of non-linear effects, or utilize any method of data interpolation.

Obviously, this type of design approach prevents any generic functional descriptions from being defined in this chapter, and as a result, their discussion will be omitted. However, Appendix II contains details concerning the calling conventions that these routines must adhere to in order to avoid modifying the control allocation routines mentioned in Section 7.1. A discussion of the functional details of these modules as they apply to the F-15 ACTIVE implementation will be presented in Chapter 8.

CHAPTER 8.

F-15 ACTIVE Implementation

This chapter describes the aircraft-specific details of Control Allocation with Rate Limiting as they apply to the F-15 ACTIVE (Advanced Control Technologies for Integrated VEhicles) research aircraft. This aircraft, which is shown in Figure 8.1, is a highly modified version of the McDonnell Douglas F-15B Fighter co-developed and supported by the NASA Dryden Flight Research Center, McDonnell Douglas Aerospace, Pratt and Whitney, and the United States Air Force as a research tool for various control law and control allocation philosophies, as well as a test platform for future thrust vectoring technologies. The aircraft is fitted with a unique quad redundant, digital, fly-by-wire flight and propulsion control system with 10 primary flight control surfaces (left/right canards, left/right ailerons, left/right stabilators, left/right rudders, and pitch/yaw thrust vectoring). For this implementation however, the rudders are used symmetrically only and are therefore combined into 1 control surface, resulting in a vector of 9 allocatable controls. It should be pointed out that the model used for this research does not include a control law, nor any equations of motion required for an aircraft simulation. It is merely used to test the control allocation algorithms using non-linear control effectiveness data representative of an actual aircraft.

Figure 8.1 The F-15 ACTIVE (Advanced Control Technologies for Integrated VEhicles)

8.1 Data Dependencies

The control effectiveness data for this particular implementation was extracted from the complete F-15 ACTIVE non-linear aerodynamic database contributed by McDonnell Douglas Aerospace using the "SweepData" utility documented in Chapter 2. The independent state parameters that the data are based on are Mach number and angle of attack for the 7 aerodynamic surfaces with an additional dependency on nozzle pressure ratio for the thrust vectoring nozzles. The sweeps were performed for a Mach range of 0.2 to 2.0 in 0.2 increments, an angle of attack range of -10 to 40 in 5 deg. increments, and nozzle pressure ratios from 1 to 25 in increments of 4. In addition, non-linearities with respect to the controls' deflections are also included. Data for these dependencies are taken from minimum position limit to maximum position limit in 5 degree increments. Other effects which are present in the complete aerodynamic database but not modeled here include flexibility effects, control/flap interaction effects, and other state dependencies such as sideslip angle, or the 3 rotational rates for the roll, pitch, and yaw axes.

Interpolation between the known node points is accomplished using the 3-D affine data interpolation technique described in Chapter 2 with Mach number, angle of attack, and control deflection as the independent lookup parameters. The 4-D lookup, (required to account for changes in nozzle pressure ratio), for the thrust vectoring effectiveness also takes advantage of the 3-D affine technique, but uses standard linear interpolation between nozzle pressure ratios. The available control effectiveness data contained in this database along with the functional dependencies are summarized in Table 8.1.

The mesh constants required for the affine data interpolation scheme and other parameters needed for control allocation are initialized by BD_ACSINIT and ACSINIT. These modules in turn call other various subroutines. After initialization, control effectiveness data for a particular axis and control is acquired through the aircraft-specific call to A_C$GETUEFF. Documentation on these subroutines can be referenced in Appendix III.

Table 8.1 Functional Dependencies for the F-15 ACTIVE Control Effectiveness Database
(N = Nozzle Pressure Ratio; M = Mach Number; α = Angle of Attack; δ = Control Deflection)

Control's Effects On…	Control Surfaces				
	Left/Right Stabilators	Left/Right Ailerons	Symmetric Rudder	Left/Right Canard	Pitch/Yaw Thrust Vectoring
Cl	(M, α, δ)	(M, α, δ)	(M, α, δ)	(M, α, δ)	(N, M, α, δ)
Cm	(M, α, δ)	(M, α, δ)	(M, α, δ)	(M, α, δ)	(N, M, α, δ)
Cn	(M, α, δ)	(M, α, δ)	(M, α, δ)	(M, α, δ)	(N, M, α, δ)
CD	(M, α, δ)	(M, α, δ)	(M, α, δ)	(M, α, δ)	(N, M, α, δ)

8.2 Control Surface Position and Rate Limits

Control surface position and rate limits may not be representative of the actual aircraft configuration since some of this information was not officially released for this research. For the aerodynamic controls and thrust vectoring nozzles, rate limits are taken as the unloaded maximum attainable rates and are assumed to remain constant regardless of the flight condition. Position limits for the aerodynamic surfaces are also taken as the unloaded limits. However, the position limits on the thrust vectoring nozzles are constrained such that the radial force exerted by each engine on the airframe structure remains below 4000 lbs. The no-load position and rate limits for the F-15 ACTIVE controls are shown in Table 8.2.

In addition, Control Allocation with Rate Limiting has the ability to compensate for actuator dynamics by over driving the allocated changes in control commands. This gain assumes a first order actuator model and requires the time constants for each control, and the sample period of the control law. For this implementation, all controls have a time constant of .0495 sec. The sample rate is assumed to be 80 Hz.

Table 8.2 Nominal Position and Rate Limits for the F-15 ACTIVE Control Surfaces

Controls	Position Limits (deg.)	Rate Limits (deg./sec.)
Left/Right Stabilators	-29,+15	±45
Left/Right Ailerons	±20	±90
Symmetric Rudder	±30	±135
Left/Right Canards	-35,+15	±75
Axisymmetric Thrust Vectoring	±20*	±80
* These limits may be reduced as a function of thrust		

8.3 Thrust Vectoring Limits

Recall from Chapter 2 that one of the assumptions in direct control allocation theory is that any two controls are uncoupled of each other, meaning that one control can be held constant while the other is free to move between its minimum and maximum deflections. It was also demonstrated in Chapter 2, that certain control allocation philosophies, such as commanding differential and symmetric controls, resulted in an admissible control subset which did not have this assumed characteristic. In the differential/symmetric allocation scheme for instance, any two related controls (ie. the left and right ailerons), produce a diamond-shaped subset (as shown in Figure 2.2) when the constraints are imposed. Under this circumstance, the two control deflection coordinates are coupled since both surfaces are required to move simultaneously along a constraint.

The mechanics of the F-15 ACTIVE thrust vectoring system produce a similar problem. One of the unique aspects of the thrust vectoring implementation on this aircraft is that the engines are equipped with an axisymmetric, Pitch/Yaw Balance Beam nozzle configuration allowing as much as 20 degrees of nozzle deflection in any direction. This type of configuration results in a rather unusual aircraft control in that it does not necessarily rotate within the aircraft body axis systems. In this implementation, the thrust vectoring nozzles are treated as two distinct controls: a pitch nozzle deflection δp, and a yaw nozzle deflection δy. It is therefore necessary to define these controls in terms of the axes about which they deflect.

Define an orthogonal thrust-axis frame F_T as follows: x_T points in the direction of the current thrust line, z_T lies in the aircraft plane of symmetry and points "down", y_T completes the right-handed coordinate system. It is desired to rotate the thrust-axis frame into one of the body-axis systems (preferably the one whose x-axis lines up with the unmodified F-15 thrust line, denoted here as F_{T0}). This requires the following sequence of rotations:

1.) Rotate F_T about the z_T-axis through the angle $-\delta y$ to bring x_T into the aircraft plane of symmetry. Call this intermediate axis-system F'.

2.) Rotate F' about the y'-axis through the angle $-\delta p$, completing the sequence of rotations.

Thus, for a given yaw and pitch nozzle configuration, the thrust forces along the x, y, and z axes (in frame F_{T0}) are found according to:

$$\begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix} = \begin{bmatrix} \cos(\delta p)\cos(\delta y) & -\cos(\delta p)\sin(\delta y) & \sin(\delta p) \\ \sin(\delta y) & \cos(\delta p) & 0 \\ -\sin(\delta p)\cos(\delta y) & \sin(\delta p)\sin(\delta y) & \cos(\delta p) \end{bmatrix} \begin{pmatrix} T \\ 0 \\ 0 \end{pmatrix} \tag{8.1}$$

Using the conventions specified above, the zero-thrust subset of admissible pitch and yaw deflections for the axisymmetric nozzles then comprise a circle with a radius of 20 degrees.

The airframe structure has been modified to accommodate any additional lateral forces produced by the off-axis engine thrust up to 4000 lbs. This structural constraint obviously requires that the radius of the previously mentioned circle change as a function of engine thrust, and offers additional complications to the control allocation algorithms. Since this circular-shaped subset of admissible controls does not produce an uncoupled pitch/yaw nozzle configuration, it must be redefined before control allocation can allocate the respective controls. In addition, it must also account for the amount of engine thrust produced and be enlarged or reduced as required. This section will therefore discuss the necessary position limit calculations performed in A_C$GETCSTR in order to produce a linearly independent subset of controls.

Under low thrust conditions, where the 20 degree maximum nozzle deflection does not exceed the lateral force limit of 4000 lbs, all that is required is to find a subset of nozzle deflections having the uncoupled properties presented in Chapter 2. This procedure is done by imposing a set of box constraints inside the circle as shown in Figure 8.2.

The problem with this method however, is that not all of the available nozzle deflections can be acquired. Therefore, a prioritization factor, $F \geq 0$, can be specified to give more or less control deflection capability in either the pitch or yaw directions. This factor is defined as the ratio of available pitch deflection to yaw deflection and determines the dimensions of the prescribed set of box constraints.

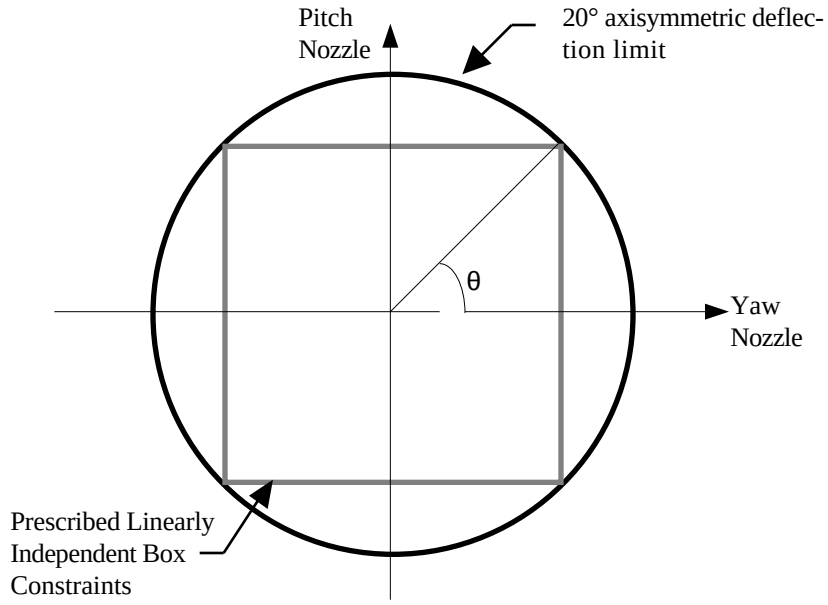

Figure 8.2 Imposed Constraints on the F-15 ACTIVE Thrust Vectoring Nozzles
This figure represents the method of imposing box constraints inside the circle of admissible nozzle deflections. The angle θ is calculated from the prioritization factor and determines the "height" and "width" of the box.

The constraints for a particular axisymmetric deflection limit δ_{max}, and prioritization factor F, are found as follows:

1.) Find the angle from the origin to the corner of the box constraints, θ, using

$$\theta = \tan^{-1}(F) \tag{8.2}$$

2.) The available pitch (δp) and yaw (δy) nozzle constraints are found according to:

$$\delta p_{max,min} = \pm \delta_{max} * \sin(\theta) \tag{8.3a}$$

$$\delta y_{max,min} = \pm \delta_{max} * \cos(\theta) \tag{8.3b}$$

Under large thrust conditions, these 2 steps are preceded by an algorithm which determines the maximum axisymmetric nozzle deflection.

The radial force exerted by the engine nozzles can be found from Eq. 8.1 and is expressed by:

$$Tr^2 = T^2 \sin^2(\delta y) + T^2 \cos^2(\delta y) \sin^2(\delta p) \tag{8.4}$$

where T is the total engine thrust. To avoid possible violation of the 4000 lb. radial limit (Tr_{max}), requires that:

$$T^2 \sin^2(\delta y) + T^2 \cos^2(\delta y) \sin^2(\delta p) \le Tr_{max}^2 \tag{8.5}$$

Equation 8.5 can then be reduced to:

$$\sin^2(\delta y) + \cos^2(\delta y) \sin^2(\delta p) \le \frac{Tr_{max}^2}{T^2} \tag{8.6}$$

Now, using the fact that the reduced subset of admissible deflections will still be a circle, either δp or δy can be set to zero, so that the maximum allowable δ_{max} can be substituted into the equation, leaving:

$$\sin^2(\delta_{max}) \le \frac{Tr_{max}^2}{T^2} \tag{8.7}$$

and the maximum allowable axisymmetric deflection can be found by:

$$\delta_{max} = \sin^{-1}\left(\frac{Tr_{max}}{T}\right) \tag{8.8}$$

The 2 steps highlighted previously can then be carried out using the maximum allowable deflection calculated in Eq. 8.8.

It should be pointed out that these techniques define the global (actual deflection) limits only, and the limits imposed by the nozzle rate capabilities remain unchanged. Documentation of the A_C$GETCSTR module that performs these steps can be referenced in Appendix III.

CHAPTER 9.

Preliminary Timing Statistics

As a conclusion to this research, some preliminary timing investigations of Control Allocation with Rate Limiting will be presented. The ultimate goal is to have the control allocation algorithms operating within a real flight control computer running at 80 Hz. It is therefore assumed that within the 1/80 sec. window, the computer must finish all of the control law calculations, invoke control allocation, and output a vector of control deflection commands.

Unfortunately, calculating the time required to allocate controls is not a straight forward task. Recall that the allocation process requires a search of each facet on the AMS until a valid intersection of the moment vector with a facet is found. Depending on the direction in moment space that the desired moments point toward, the valid facet could be either the first checked or the last. As a result, the time required to allocate is not constant from one frame to another. It may be possible to implement a “smarter” search algorithm that would allow control allocation to start searching facets which lie in the vicinity of the desired moment vector first. This method would have the benefit of decreasing the time required to allocate significantly, yet, as of now, no algorithm of this type has been found. The current implementation however, begins searching the facet that produced results in the previous frame. The idea for this type of search is based on the fact that the overall control configuration will not change considerably from one frame to another. Of course, there are some exceptions to this rule as well. It is not hard to imagine a maneuver that traverses along the edge of two facets such that the intersection jumps between them, leaving the control allocator with the task of starting a new search every frame.

Because of the current facet searching scheme, the control allocation timing is heavily dependent on the maneuver chosen. A smooth set of moment time histories that point in the same general direction in moment space will result in better timing statistics than a set of

moment time histories which change directions very frequently. As a result, the question of which type of maneuver to perform when gathering timing data is not an easy one to answer. For this research, it was decided to provide random moment inputs for a 10 sec. time duration. Although a timing analysis of this nature may not be a fair estimate of the true speed of control allocation, it is assumed to provide a worst case scenario in which practically every sample frame requires control allocation to invoke a new search.

9.1 Timing Results

Timing statistics are taken from a 66 MHz PowerPC 601 RISC computer system, running the simulation at 80 Hz. Thus, if the amount of computational time required by the control laws is assumed to be negligible, then the maximum time available to CARL for allocating controls without missing a sample frame is 0.0125 sec. Of course, the amount of time required by the control laws is usually quite significant, and is generally a function of the type of flight control computer implemented, and the complexity of the control law algorithms. For the timing test presented here, however, it was decided to allocate half of the sample period to the control law and the remaining half to CARL, resulting in an acceptable time frame between 0 and 6.25 msec.

The 9 controls of the F-15 ACTIVE control allocation model result in a minimum of 72 facets that may have to be searched in any given frame. (Note that whenever redundant controls are present, not all of the 72 facets can be directly determined and the control allocation algorithms will have to check some of the interior faces as well). Minimum norm restoring was also enabled.

Elapsed time was calculated through a call to an operating system function that returns the time since startup, quantized to 1/60 sec. intervals. Thus, the results had to be partitioned into 4 possible time ranges, each bracketed by an odd multiple of 1/120 of a second. These are shown in Figure 9.1. Out of the 801 iterations performed during this worst case simulation, control allocation was only able to allocate in an acceptable time frame (the 0 - 0.0083 sec. range) 15.4% of the time (about 123 iterations).

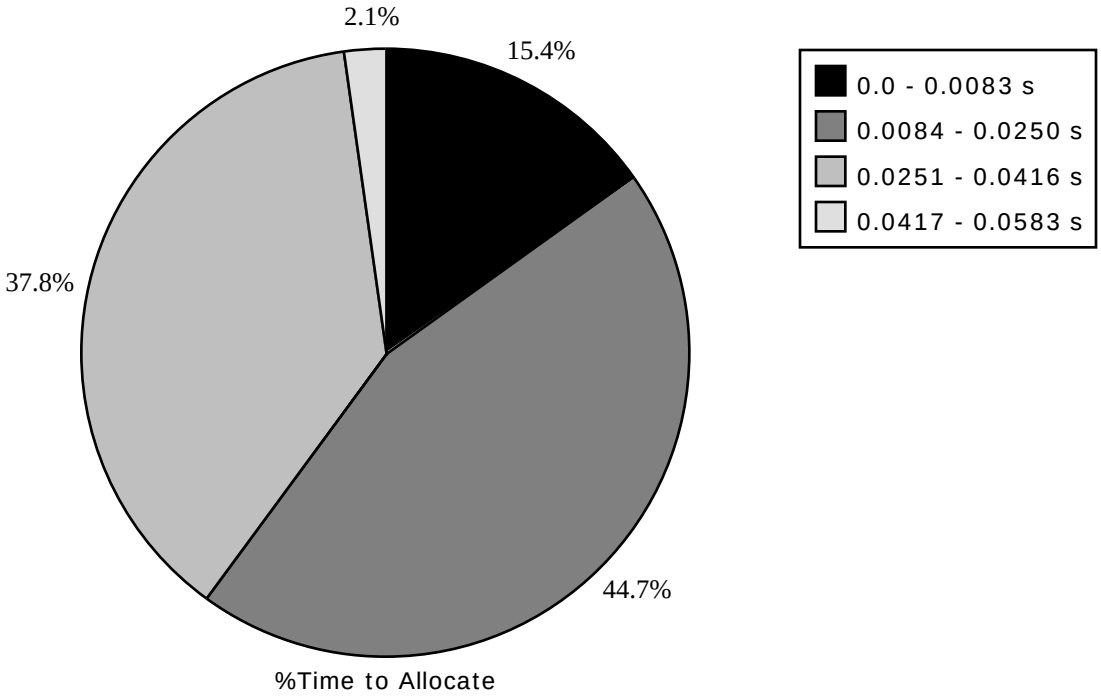

Figure 9.1 Control Allocation with Rate Limiting: Timing Statistics
This pie chart represents the amount of time needed to allocate controls and the recurrence of each time range. Percentages are based on random vectors of moments for a 10 second simulation run at 80 Hz. Minimum Norm restoring is enabled.

Based on the timing results shown in Figure 9.1, this test indicates either a need for faster, more powerful computers, or consideration of a parallel processing implementation. Since the greatest amount of time is spent by control allocation searching facets, which simply requires the same set of calculations to be performed repeatedly, a parallel processing implementation may be the best option. In addition, the calculations performed for each facet search are nothing more than a few floating point additions and multiplications, so that the processors would not have to be extremely powerful.

Since the bounding facets of the AMS always occur in pairs (ie. think of facets as being the front and back faces of a cube), the proposed device would only need one processor per every two facets. Thus, the F-15 ACTIVE device would consist of 36 processors. The need for a separate “Control Allocator” computer module may increase the costs required to implement such techniques into an aircraft. However, an implementation of this type would also introduce several benefits. First of all, recall from Figure 9.1 that the time required to allocate varied considerably, depending on the order in which the facets were checked. The time required to allocate controls using a multiprocessing, Control Allocator Box would be constant, so that there would no longer be any concerns for the effects of variable transport delays. Second, all the benefits of constrained control allocation would be available to the aircraft. These include all of the moment rate capabilities of Control Allocation with Rate Limiting, various control restoring algorithms such as minimum drag control allocation, and the ability to reconfigure controls in the event of failures.

CHAPTER 10.

Conclusions

The direct determination of controls in the constrained control allocation problem is computationally more complicated than other control allocation methods. This complexity is due to the geometrical principles involved, such as defining the boundaries of the Attainable Moment Subset, and finding the intersection of the desired moment vector with one of the bounding facets. However, once the intersection is found, determining the required control deflections becomes straight forward. By understanding the theory behind these direct methods, the advantages of using them becomes quite clear. Since they guarantee to use the maximum moment-generating capabilities of the control surfaces, they can be used to evaluate the efficiency of other control allocation methods. These utilities would then allow the designer or contractor to determine for a particular design, whether additional control capabilities are required, or whether existing controls offer little capabilities and may be eliminated. Additionally, these techniques could be integrated into current aircraft designs, that may not utilize the controls as efficiently as possible, to produce a more maneuverable aircraft.

The focus of this research has been aimed toward the implementation of Constrained Control Allocation techniques into current aircraft configurations. The following conclusions and recommendations have been drawn:

Due to the uncoupled control surface requirements of Constrained Control Allocation, the best (and preferred) method of gathering control effectiveness data is in terms of left and right control surfaces as opposed to symmetric and differential deflections. In situations where a set of uncoupled constraints does not exist, (such as the F-15 ACTIVE thrust vectoring nozzles), an algorithm may need to be employed to transform the set of admissible controls into a form more suitable for control allocation.

Constrained Control Allocation can be implemented in a discrete time domain to

allocate deflection rates in response to commanded moment rates. This type of allocation scheme, known as Control Allocation with Rate Limiting (CARL), has the advantage of being able to include nonlinear control effects, and to allocate controls without violating any position or rate limits. One of the drawbacks to this scheme however, is the control wind-up problem that results from the path-dependent nature of the discrete algorithm. This problem must be alleviated by applying some type of control restoring technique. Any optimization objective can be defined in the restoring problem provided that it can be expressed in terms of control deflections. Restoring techniques investigated so far include minimum-norm restoring and minimum-drag restoring.

One of the advantages of control allocation schemes in general is the ability to reconfigure controls in response to a detected failure. The “delta” moment allocation scheme used by CARL builds upon this idea by introducing a method of canceling the errors associated with these failed controls.

When using discrete algorithms like CARL that allocate changes in controls for every sample frame, careful attention must be paid to the sample period and how it relates to the controls’ actuator dynamics. If the sample period is suitably “small”, then the actuator dynamics may play a significant role in the allocation process and may have to be corrected through some type of compensation. In this research, a vector of gains is found, based on a first order actuator model, to overdrive the allocated changes in controls so that the actuator responses match the desired responses more accurately.

Finally, when allocating controls, these algorithms have to search each facet on the boundary of the Attainable Moment Subset in series. The time required to allocate then varies considerably and depends on the order in which the facets are checked. For this reason, it may be very beneficial to investigate the possibilities of a multi-processing implementation in which each processor is responsible for a small, predetermined number of facets to check.

REFERENCES

1. Durham, W.C., "Constrained Control Allocation," *Journal of Guidance, Control, and Dynamics,* Vol. 16, No. 4, 1993, pp. 717-725.

2. Durham, W.C., "Constrained Control Allocation: Three-Moment Problem," *Journal of Guidance, Control, and Dynamics,* Vol. 17, No. 2, 1993, pp. 330-336.

3. Durham, W.C., "Attainable Moments for the Constrained Control Allocation Problem," *Journal of Guidance, Control, and Dynamics,* Vol. 17, No. 6, 1994, pp. 1371-1373.

4. Durham, W.C. and Bordignon, K.A., "Multiple Control Effector Rate Limiting," *Journal of Guidance, Control and Dynamics,* Vol. 19, No. 1, 1995, pp. 30-37.

5. Durham, W.C. Bolling, J.G., and Bordignon, K.A., "Minimum Drag Control Allocation," *Journal of Guidance, Control, and Dynamics,* Vol. 20, No. 1, 1997, pp. 190-193.

6. Durham, W., Bolling, J., and Hann, C., "Simulator Implmentation of Direct Control Allocation Methods," *Proceedings of the AIAA Flight Simulation Technologies Conference* (Baltimore, MD), AIAA, Washington, DC, 1995 (AIAA Paper 95-3380).

7. Sinnet, M.K., Steck, J.E., Selbert, B.P., and Oetting, R.B., "An Alternate Table Look-Up Routine for Real-Time Digital Flight Simulation," *Proceedings of the AIAA Flight Simulation Technologies Conference* (Boston, MA), AIAA Washington, DC, 1989 (AIAA Paper 89-3310).

8. *Matlab External Interface Guide,* The Mathworks Inc., Natick, Mass., 1995.

9. Dornheim, M. A., "Report Pinpoints Factors Leading to YF-22 Crash," *Aviation Week and Space Technology,* Vol. 137, Nov. 1992, pp. 53,54.

10. Shifrin, C. A., "Sweden Seeks Cause of Gripen Crash," *Aviation Week and Space Technology,* Vol. 139, Aug. 1993, pp. 78,79.

11. McKay, K., "Summary of an AGARD Workshop on Pilot Induced Oscillation," *Proceedings of the AIAA Guidance, Navigation, and Control Conference* (Scottsdale, AZ), AIAA, Washington, DC, 1994, (AIAA Paper 94-3668).

12. A'Harrah, R., "An Alternate Control Scheme for Alleviating Aircraft Pilot Coupling," *Proceedings of the AIAA Guidance Navigation and Control Conference* (Scottsdale, AZ), AIAA, Washington, DC, 1994, (AIAA Paper 94-3673).

13. Franklin, G., Powell, J., and Workman, M., *Digital Control of Dynamic Systems,* 2nd Edition, Addison Wesley Publishing Company, New York, 1990.

14. Brogan, W. L., *Modern Control Theory,* 2nd Edition, Prentice-Hall, Englewood Cliffs, NJ, 1982.

15. Raymond, E.T. and Chenoweth, C.C., *Aircraft Flight Control Actuation System Design,* Society of Automotive Engineers, Inc., Warrendale, PA, 1993.

16. Stevens, B.L. and Lewis, F.L., *Aircraft Control and Simulation,* John Wiley and Sons, Inc., New York, 1992.

APPENDIX I.

Data Collection Utilities

This appendix contains detailed descriptions of the utilities required to extract effectiveness data from an aircraft database, generate the mesh constants for a given table using the affine data interpolation technique, and to convert the data to a readable format.

1. The Sweep Data Utility

A. Usage

This FORTRAN application can be linked to a given aerodynamic database and used to extract effectiveness data or data increments for a wide range of independent and dependent parameters, and to store the extracted data as Matlab workspace (*.mat) files. It has been designed as a stand-alone program and uses a convenient command line interface library. Additional static libraries required are the Matlab external interface libraries, and an aircraft specific library containing the necessary aerodynamic lookup routines.

A.1 LAUNCH_SD

Function Prototype, PROGRAM
COMMON SIMPARS, SHELLPARMS, SWEEPPARMS, UEFFECTS

This is the main PROGRAM unit required to launch the Sweep Data utility.

LAUNCH_SD

LAUNCH_SD initializes the necessary variables required for the Shell interface, initializes the Sweep Data program specific parameters, and initializes the aerodynamic table lookup routines

Global Definitions

SIMPARS	[global]	Contains the “Simulation Shell” parameters. Some of these are required by the Shell Interface library.
SHELLPARMS	[global]	Contains additional Shell interface parameters
SWEEPPARMS	[global]	Contains global variables specific to the Sweep Data utility
UEFFECTS	[global]	Contains intermediate results calculated in the aerodynamic database

B. General Remarks

The SIMPARS common block contains two arrays; a 30 element LOGICAL array and a 10 element CHARACTER*80 array. For the Sweep Data utility, only two of these variables are required. These are the INITIALIZED flag (SIMPARL(8)) and the DO_DIAGS flag (SIMPARL(1)). The SHELLPARMS common block is required by the Shell interface code. The CALLERID and MODE variables within this common should be set to 0 and 1 at launch time. The other variables are handled by the Shell interfaces and should not be tampered with. The

contents of the SWEEPPARMS common block are described below. NP represents the number of independent parameters that the user is allowed to sweep, and NC represents the number of dependent aerodynamic coefficients that the user can record during a sweep.

Variable	Type	Description
PARM(NP)	REAL*8	Stores the current values for all "NP" independent parameters
PARMNAME(NP)	CHARACTER*8	The name of each independent parameter
PARMDSCRP(NP)	CHARACTER*32	A brief description of each independent parameter
PARMMIN(NP)	REAL*8	The minimum possible value for each ind. parameter
PARMMAX(NP)	REAL*8	The maximum possible value for each ind. parameter
PARMINC(NP)	REAL*8	The amount to increment each ind. parameter by
COEFF(NP,NC)	REAL*8	Stores the total change of the "NC" dependent variable associated with the "NP" independent variable
COEFFNAME(NC)	CHARACTER*8	The name of each dependent variable
COEFFDSCRP(NC)	CHARACTER*32	A brief description of each dependent variable

The contents of the UEFFECTS common block contain intermediate results gathered in the aerodynamic database and are optional. For instance, if it is desired to store the pitching moment increments due to the right stabilator only (no other interactions like flexibility effects), then this intermediate result would be stored in this common block. UEFFECTS is defined by:

Variable	Type	Description
UEFF(NP,NC)	REAL*8	Stores the change of the "NC" dependent variable associated with the "NP" independent variable only (no other interactions)
UEFFNAME(NP,NC)	CHARACTER*8	The name of the intermediate result associated with the NP independent parameter and the NC dependent variable

C. Functional Description

As previously mentioned, this module serves only to initialize the required elements when the Sweep Data utility is launched. The first call is a MacOS system call to OUTWINDOWSCROLL to set up the scrolling properties for the applications Shell window. Next, BDSWEEPDATA is called to initialize the independent and dependent variable names, minimum values, maximum values, and descriptions for the common blocks SWEEPPARMS and UEFFECTS. After the call to BDSWEEPDATA, some of the Shell interface parameters are initialized. INITIALIZED is set to FALSE, indicating that the application has not yet initialized itself, CALLERID is set to 0 (indicating to the shell interface libraries that the user interface is in command line mode), MODE is set to 1 (This effects the way the Sweep Data shell reacts to a carriage return at the command prompt), and the Command buffer is cleared by setting COMMANDBUFFER to a blank string. The next section initializes the aerodynamic lookup routines. Although the code in this section can change depending on the database that is being used, the developer should ensure that when the database is called from this module it is initialized. (This can be done by including the INITIALIZED flag in the argument list to the database lookup routine). Finally, SHELL_GLOBALS and SHELL_SWEEPDATA are called to initialize the global and "Sweep Data" valid command records. After initialization, INITIALIZED is set to TRUE and SHELL_SWEEPDATA is called to start handling user commands.

D. Errors and Restrictions

This utility has been successfully compiled and tested on both 68040 Macintosh computers and PowerPC Macintosh

computers using the MPW 3.4.2 programming environment and the Language Systems FORTRAN compiler. It is not guaranteed that the code as it exists, will run, or even compile on other machines without errors.

Be aware that the Shell Interface libraries contain some Macintosh Toolbox calls that are used to display dialog boxes and windows when the CALLERID flag is set to 1. This utility does not support the use of a graphical frontend, and errors will certainly occur if CALLERID is not 0. In addition, indications of an improper MODE setting may include program quitting every time the carriage return is pressed at the command prompt. To prevent this, make sure that MODE is always 1. This simply causes the Shell code to redraw the menu without exiting.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: Launch_SD
!           Called By: none
!           Calls to: BDSWEEPDATA, SHELL_SWEEPDATA, F15AERO,
!                     SHELL_GLOBALS
!
! ------------------------------------------------------------------------
      PROGRAM Launch_SD
! ------------------------------------------------------------------------
!
!       Function:   Sets up the SweepData application if it is launched
!                   seperately from the Simulation Shell.
!
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                         By
!  JUN 10 1996        Created                                J. Bolling
!
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------
      LOGICAL SIMPARL( 15)
      CHARACTER*80 SIMPARC80(3)

      LOGICAL Initialized
! ----------------------------Shell Parameters-------------------------
      INTEGER CallerID,Mode
      LOGICAL Do_Menu,Did_Menu
      CHARACTER*80 CommandBuffer
! ----------------------------Sweep Parameters-------------------------
      INTEGER NParms,NCoeffs
```

```
      PARAMETER (NParms = 29, NCoeffs = 6)
      DOUBLE PRECISION PARM(NParms), PARMMIN(NParms), PARMMAX(NParms),
     .                 COEFF(NParms,NCoeffs), UEFF(NParms,NCoeffs),
     .                 PARMINC(3)
      CHARACTER*8 PARMNAME(NParms), COEFFNAME(NCoeffs),
     .            UEFFNAME(NParms,NCoeffs)
      CHARACTER*32 PARMDSCRP(NParms), COEFFDSCRP(NCoeffs)
! ---------------------------Locals-----------------------------------
      REAL CDT, CYT, CLT, CLLT, CMT, CNT, VT, ALFA, XMACH, QBAR, P, Q,
     .     R, ALT, BETA, NPRNR4, LEXA, REXA
      LOGICAL Do_COLLECTIVE
      DOUBLE PRECISION CSMODE
! ----------------------------------------------------------------------
!
!     Common Section
!
! ----------------------------------------------------------------------
      COMMON / SWEEPPARMS / PARM, PARMNAME, PARMDSCRP, PARMMIN, PARMMAX,
     .                      PARMINC, COEFF, COEFFNAME, COEFFDSCRP
      COMMON / UEFFECTS / UEFF, UEFFNAME
      COMMON / SHELLPARMS / CallerID, Mode, Do_Menu, Did_Menu,
     .                      CommandBuffer
      COMMON / SIMPARS / SIMPARL,SIMPARC80
! ----------------------------------------------------------------------
!
!     Equivalence Section
!
! ----------------------------------------------------------------------
      EQUIVALENCE (SIMPARL(8)    , Initialized    )
! ----------------------------------------------------------------------
!
!     Initialization Section
!
! ----------------------------------------------------------------------
      CALL OutWindowScroll(9999)

! Set up Data Sweep specific items here

      CALL BDSWEEPDATA

      Initialized = .FALSE.
      CallerID = 0
      Mode = 1
      CommandBuffer = ' '

! Initialize the database

      Do_COLLECTIVE = .FALSE.
      CSMODE = 1.0

      CALL F15AERO(CDT, CLT, CMT, CYT, CNT, CLLT, ALFA, XMACH, QBAR,
```

```
     .             VT,  P,   Q,   R,   ALT, 0.0,  BETA, 0.0,   0.0,
     .             0.0, NPRNR4, LEXA, REXA, CSMODE, Initialized,
     .             Do_COLLECTIVE)

! Initialize SHELL_GLOBALS and the SweepData Shell

      CALL SHELL_GLOBALS
      CALL SHELL_SWEEPDATA

      Initialized = .TRUE.
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      CALL SHELL_SWEEPDATA
! ---------------------------------------------------------------------
!
!     End Of DataSweep
!
! ---------------------------------------------------------------------
      END
```

APPENDIX I.

1.1 Sweep Data Initialization and the Shell Interface

A. Usage

This section describes the subroutines required to initialize and handle execution of the Sweep Data utility

A.1 BDSWEEPDATA

Function Prototype, BLOCK DATA

COMMON SWEEPPARMS, UEFFECTS

Use this BLOCK DATA module to initialize all of the independent and dependent variable names, limits, and descriptions, as well as the names of any intermediate results (if used).

CALL BDSWEEPDATA

BDSWEEPDATA contains no executable statements and only initializes the data in common blocks SWEEPPARMS and UEFFECTS.

Global Definitions

SWEEPPARMS	[global]	Contains global variables specific to the Sweep Data utility
UEFFECTS	[global]	Contains intermediate results calculated in the aerodynamic database

A.2 SHELL_SWEEPDATA

Function Prototype, SUBROUTINE

COMMON SHELLPARMS, SWEEPPARMS, SIMPARS, DSMENU

This subroutine handles user commands for the Sweep Data shell interface

CALL SHELL_SWEEPDATA

SHELL_SWEEPDATA displays the Sweep Data command menu, displays a command prompt, and waits for user response.

Global Definitions

SWEEPPARMS	[global]	Contains global variables specific to the Sweep Data utility
SIMPARS	[global]	Contains the “Simulation Shell” parameters. Some of these are required by the Shell Interface library.
SHELLPARMS	[global]	Contains additional Shell interface parameters
DSMENU	[global]	Contains the valid command record for the Sweep Data interface

B. General Remarks

Since the structure of aerodynamic databases varies significantly for different aircraft models, the BDSWEEPDATA module has been designed to provide a standard way of defining the available independent parameters, their descriptions, and their maximum and minimum allowable values for any table lookup architecture. All the developer needs to do after creating this module, is to assign (or EQUIVALENCE) the desired parameters within the table

lookup procedures, and within the SWEEPDATA routine itself.

The SHELL_SWEEPDATA event loop acts as the interface shell to the SWEEPDATA algorithm and depends heavily on the parameters defined in BDSWEEPDATA.

C. Functional Description

BDSWEEPDATA has no executable statements.

After initialization, SHELL_SWEEPDATA will start an “event loop” in which it calls GETMENUTEXT and GETMENUCOMMAND, to display the list of commands and wait for user response. When GETMENUCOMMAND receives a valid command it returns the command record index in the STATEMENT field. SHELL_SWEEPDATA then performs the following actions, and then repeats the loop:

STATEMENT = 1; SHELL_SWEEPDATA toggles the DO_COLLECTIVE flag (allows the use of symmetric and differential control deflections instead of right and left deflections)
STATEMENT = 2; SHELL_SWEEPDATA toggles the DO_TOTAL flag (Results obtained from the datasweep are the total coefficients returned from the database instead of intermediate results)
STATEMENT = 3; SHELL_SWEEPDATA toggles the DO_FLAPSCD flag (Allows leading and trailing edge flap schedules to override current flap positions)
STATEMENT = 4; SHELL_SWEEPDATA toggles the DCDU derivative extraction flag (results returned from datasweep are derivatives w.r.t the last independent parameter instead of increments)
STATEMENT = 5; SHELL_SWEEPDATA requests the names of the independent parameters to sweep.
STATEMENT = 6; SHELL_SWEEPDATA requests the names of the dependent variables to save.
STATEMENT = 7; SHELL_SWEEPDATA requests the name of an independent parameter to set and hold constant.
STATEMENT = 8; SHELL_SWEEPDATA requests the amount to increment each specified independent parameter for a data sweep.
STATEMENT = 9; SHELL_SWEEPDATA calls the SWEEPDATA algorithm to perform a data sweep using the specified independent parameters and increments.
STATEMENT = 10; SHELL_SWEEPDATA displays the available independent sweep parameters, dependent variables, and their descriptions.
STATEMENT = 11; SHELL_SWEEPDATA sets all independent parameters to 0
STATEMENT = 12; (a global command was entered) SHELL_SWEEPDATA does nothing
STATEMENT = 13; (carriage return was pressed at the command prompt) SHELL_SWEEPDATA redraws the menu

D. Errors and Restrictions

The Shell Interface libraries are in their 3rd year of development and are relatively stable. Two known limitations still exist however. When typing multiple commands on one line, separate them by a SPACE. Other characters cause invalid command errors. Second, when entering numerical information (for instance, the SET command), make sure that the data entered is the same type as the data expected. Entering ascii characters when the interface expects a numerical value often results in an IO error.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ----------------------------------------------------------------------
!
!       Module Name: BDSWEEPDATA
```

```
!           Called By: Launch_SD
!           Calls to: none
!
! ----------------------------------------------------------------------
      BLOCK DATA BDSWEEPDATA
! ----------------------------------------------------------------------
!
!       Function:   Block Data module for the Sweep Data utility. The
!                 purpose of this module is to initialize the independent
!                 and dependent parameter names and descriptions for the
!                 particular database to be swept. This module is cur-
!                 rently written to support the F15 ACTIVE database
!                 F15AERO.
!
!
! ----------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                          By
!  JUN 10 1996        Created                                J. Bolling
!
! ----------------------------------------------------------------------
      IMPLICIT NONE
! ----------------------------------------------------------------------
!
!     Declaration Section
!
! ----------------------------------------------------------------------
      INTEGER NParms,NCoeffs
      PARAMETER (NParms = 29, NCoeffs = 6)
      DOUBLE PRECISION PARM(NParms), PARMMIN(NParms), PARMMAX(NParms),
     .                 COEFF(NParms,NCoeffs), UEFF(NParms,NCoeffs),
     .                 PARMINC(3)
      CHARACTER*8 PARMNAME(NParms), COEFFNAME(NCoeffs),
     .            UEFFNAME(NParms,NCoeffs)
      CHARACTER*32 PARMDSCRP(NParms), COEFFDSCRP(NCoeffs)
      INTEGER I,J
! ----------------------------------------------------------------------
!
!     Common Section
!
! ----------------------------------------------------------------------
      COMMON / SWEEPPARMS / PARM, PARMNAME, PARMDSCRP, PARMMIN, PARMMAX,
     .                      PARMINC, COEFF, COEFFNAME, COEFFDSCRP
      COMMON / UEFFECTS / UEFF, UEFFNAME
! ----------------------------------------------------------------------
!
!     Data Section
!
! ----------------------------------------------------------------------
      DATA PARMNAME   /
```

```
1  'ALPHA   ','MACH    ','BETA    ','P       ','Q       ',
2  'R       ','ALT     ','NPR     ','DCL     ','DCR     ',
3  'DAL     ','DAR     ','TEFL    ','TEFR    ','DRL     ',
4  'DRR     ','DHTL    ','DHTR    ','DNOZP   ','DNOZY   ',
5  'DCANRD  ','DDCANRD ','DRAILD  ','DAILD   ','DTEFD   ',
6  'DDTEF   ','DSTBD   ','DTALD   ','DRUDD   '
.                /
 DATA PARMMIN    /
1   -10.0    ,  0.2     , -20.0    , -6.28   , -6.28     ,
2   -6.28    ,  0.0     ,  1.0     , -35.0   , -35.0     ,
3   -20.0    , -20.0    ,  0.0     ,  0.0    , -30.0     ,
4   -30.0    , -30.0    , -30.0    , -20.0   , -20.0     ,
5   -35.0    , -15.0    , -20.0    , -40.0   ,   0.0     ,
6     0.0    , -30.0    , -45.0    , -30.0
.                /
 DATA PARMMAX    /
1    40.0    ,  2.0     ,  20.0    ,  6.28   ,  6.28     ,
2     6.28   , 50000.0  ,  25.0    , 15.0    , 15.0      ,
3    20.0    ,  20.0    ,  20.0    , 20.0    , 30.0      ,
4    30.0    ,  15.0    ,  15.0    , 20.0    , 20.0      ,
5    15.0    ,  15.0    ,  20.0    , 40.0    , 20.0      ,
6     0.0    ,  15.0    ,  45.0    , 30.0
.                /
 DATA PARMINC    /
.     0.0    ,   0.0    ,   0.0
.                /
 DATA PARMDSCRP  /
1  'Angle of attack (deg)',
2  'Mach number (nd)',
3  'Sideslip angle (deg)',
4  'Body axis Roll Rate (rad/sec)',
5  'Body axis pitch rate (rad/sec)',
6  'Body axis yaw rate (rad/sec)',
7  'Altitude (ft)',
8  'Nozzle pressure ratio (nd)',
9  'Left Canard deflection (deg)',
.  'Right Canard deflection (deg)',
1  'Left Aileron deflection (deg)',
2  'Right Aileron deflection (deg)',
3  'Left T.E. Flap deflection (deg)',
4  'Right T.E. Flap deflection (deg)',
5  'Left Rudder deflection (Deg)',
6  'Right Rudder deflection (deg)',
7  'Left stabilator deflection (deg)',
8  'Right stabilator def. (deg)',
9  'Pitch Nozzle Deflection (deg)',
.  'Yaw Nozzle Deflection (deg)',
1  'Symmetric Canards (deg)',
2  'Differential Canards (deg)',
3  'Symmetric ailerons (deg)',
4  'Differential ailerons (deg)',
```

```
     5  'Symmetric T.E.F. (deg)',
     6  'Differential T.E.F. (Deg)',
     7  'Symmetric stabilator (deg)',
     8  'Differnetial stabilator (deg)',
     9  'Symmetric Rudder (deg)'
     .                /
      DATA COEFFNAME  /
     .  'CD      ','CL      ','CM      ','CY      ','C1      ',
     .  'CN      '
     .                /
      DATA COEFFDSCRP /
     1  'Drag Coefficient',
     2  'Lift Coefficient',
     3  'Pitch Moment coefficent',
     4  'Sideforce coefficient',
     5  'Rolling moment coefficient',
     6  'Yawing moment coefficient'
     .                /
      DATA ((UEFFNAME(i,j),j=1,NCoeffs),i=1,NParms)   /
     1'CFX1    ','CFZ1    ','CMM1    ','CFY1    ','CML1    ','CMN1    ',
     2'CFX1    ','CFZ1    ','CMM1    ','CFY1    ','CML1    ','CMN1    ',
     3'DCXSB   ','DCZSB   ','DCMSB   ','CFY1    ','CML1    ','CMN1    ',
     4'CFX1    ','CFZ1    ','CMM1    ','CFY1    ','CMLP    ','CMNP    ',
     5'CFX1    ','CFZ1    ','CMMQ    ','CFY1    ','CML1    ','CMN1    ',
     6'CFX1    ','CFZ1    ','CMM1    ','CFY1    ','CMLR    ','CMNR    ',
     7'CFX1    ','CFZ1    ','CMM1    ','CFY1    ','CML1    ','CMN1    ',
     8'DXNOZ   ','DZNOZ   ','DMNOZ   ','DYNOZ   ','DLNOZ   ','DNNOZ   ',
     9'CXCAND  ','CZCAND  ','CMCAND  ','CYDC    ','CLDC    ','CNDC    ',
     .'CXCAND  ','CZCAND  ','CMCAND  ','CYDC    ','CLDC    ','CNDC    ',
     1'DCXDA1  ','CZTEF   ','CMTEF   ','CYDAD   ','CLDAD   ','CNDAD   ',
     2'DCXDA1  ','CZTEF   ','CMTEF   ','CYDAD   ','CLDAD   ','CNDAD   ',
     3'DCXDA1  ','CZTEF   ','CMTEF   ','CFYFLP  ','CMLFLP  ','CMNFLP  ',
     4'DCXDA1  ','CZTEF   ','CMTEF   ','CFYFLP  ','CMLFLP  ','CMNFLP  ',
     5'DCXDR1  ','CFZ1    ','DCMDR1  ','CYDRD   ','CLDRD   ','CNDRD   ',
     6'DCXDR1  ','CFZ1    ','DCMDR1  ','CYDRD   ','CLDRD   ','CNDRD   ',
     7'DCXDS1  ','DCZDS1  ','DCMDS1  ','CYDTD   ','CLDTD   ','CNDTD   ',
     8'DCXDS1  ','DCZDS1  ','DCMDS1  ','CYDTD   ','CLDTD   ','CNDTD   ',
     9'DXPNOZ  ','DZPNOZ  ','DMPNOZ  ','DYPNOZ  ','DLPNOZ  ','DNPNOZ  ',
     .'DXYNOZ  ','DZYNOZ  ','DMYNOZ  ','DYYNOZ  ','DLYNOZ  ','DNYNOZ  ',
     1'CXCAND  ','CZCAND  ','CMCAND  ','CYDC    ','CLDC    ','CNDC    ',
     2'CXCAND  ','CZCAND  ','CMCAND  ','CYDC    ','CLDC    ','CNDC    ',
     3'DCXDA1  ','CZTEF   ','CMTEF   ','CYDAD   ','CLDAD   ','CNDAD   ',
     4'DCXDA1  ','CZTEF   ','CMTEF   ','CYDAD   ','CLDAD   ','CNDAD   ',
     5'DCXDA1  ','CZTEF   ','CMTEF   ','CFYFLP  ','CMLFLP  ','CMNFLP  ',
     6'DCXDA1  ','CZTEF   ','CMTEF   ','CFYFLP  ','CMLFLP  ','CMNFLP  ',
     7'DCXDS1  ','DCZDS1  ','DCMDS1  ','CYDTD   ','CLDTD   ','CNDTD   ',
     8'DCXDS1  ','DCZDS1  ','DCMDS1  ','CYDTD   ','CLDTD   ','CNDTD   ',
     9'DCXDR1  ','CFZ1    ','DCMDR1  ','CYDRD   ','CLDRD   ','CNDRD   '
     .                                                    /

! ---------------------------------------------------------------------
```

```
!
!     End of BDSWEEPDATA
!
! ----------------------------------------------------------------------
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ----------------------------------------------------------------------
!
!       Module Name: SHELL_SWEEPDATA
!           Called By: Launch_SD
!           Calls to: SWEEPDATA, Shell library routines
!
! ----------------------------------------------------------------------
      SUBROUTINE SHELL_SWEEPDATA
! ----------------------------------------------------------------------
!
!        Function:    Performs the user interface tasks for the Sweep
!                     Data utility
!
! ----------------------------------------------------------------------
!
!        Modifications
!     Date                        Purpose                          By
!  JUN 10 1996        Created                                  J. Bolling
!
! ----------------------------------------------------------------------
      IMPLICIT NONE
! ----------------------------------------------------------------------
!
!     Declaration Section
!
! ----------------------------------------------------------------------
      LOGICAL SIMPARL( 15)
      CHARACTER*80 SIMPARC80(3)

      LOGICAL Initialized
! ----------------------------Shell Parameters--------------------------
      INTEGER CallerID,Mode
      LOGICAL Do_Menu,Did_Menu
      CHARACTER*80 CommandBuffer
! ----------------------------Sweep Parameters--------------------------
      INTEGER NParms,NCoeffs
      PARAMETER (NParms = 29, NCoeffs = 6)
      DOUBLE PRECISION PARM(NParms), PARMMIN(NParms), PARMMAX(NParms),
     .                 COEFF(NParms,NCoeffs), UEFF(NParms,NCoeffs),
     .                 PARMINC(3)
      CHARACTER*8 PARMNAME(NParms), COEFFNAME(NCoeffs),
     .            UEFFNAME(NParms,NCoeffs)
      CHARACTER*32 PARMDSCRP(NParms), COEFFDSCRP(NCoeffs)
! ----------------------------Locals------------------------------------
```

```
      INCLUDE ':INCLUDES:Cmd_Structure.txt/LIST'
      INTEGER STATEMENT, IOStat, NIP, NDP, I, J, K, IPARM(3),
     .        ICOEFF(6)
      CHARACTER*80 Message(40)
      CHARACTER*8 DSPrompt,NILPrompt,IVAR,DVAR,VARPrompt
      CHARACTER*12 VARPrompt12
      PARAMETER (DSPrompt = 'DS>', NILPrompt = ' ')
      LOGICAL Do_FLAPSCD, Do_COLLECTIVE, Do_TOTAL, IFoundit, DCDU
      REAL TEMPR4
! ----------------------------Functions-------------------------------
      INTEGER ProcessIOString
      CHARACTER*8 GetCHARInput, I4ToString
      REAL GetREALInput
      INTEGER GetINTInput
      LOGICAL GetLOGICALInput
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / SHELLPARMS / CallerID, Mode, Do_Menu, Did_Menu,
     .                      CommandBuffer
      COMMON / SWEEPPARMS / PARM, PARMNAME, PARMDSCRP, PARMMIN, PARMMAX,
     .                      PARMINC, COEFF, COEFFNAME, COEFFDSCRP
      COMMON / SIMPARS / SIMPARL,SIMPARC80
      COMMON / DSMENU  / DSCMD
! ---------------------------------------------------------------------
!
!     Equivalence Section
!
! ---------------------------------------------------------------------
      EQUIVALENCE (SIMPARL(8)    , Initialized    )
! ---------------------------------------------------------------------
!
!     Initialization Section
!
! ---------------------------------------------------------------------
      INCLUDE ':INCLUDES:SDCmd.txt/LIST'

      IF (.not. Initialized) THEN
        DO 1060 I = 1,NParms
          PARM(I) = 0.0D0
1060    CONTINUE
      END IF

      Do_FLAPSCD = .FALSE.
      Do_COLLECTIVE = .FALSE.
      Do_TOTAL = .TRUE.
      DCDU = .FALSE.
      IVAR = 'none'
      Do_Menu = .TRUE.
```

```
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------

      IF (Initialized) THEN

100   CALL GetMenuText(DSCMD)
      CALL GetMenuCommand(DSCMD,DSPrompt,STATEMENT)

      GO TO (40,15,30,50,10,20,25,55,60,80,85,100,100) STATEMENT

10    CONTINUE                 ! Set Independent Variables to sweep
        NIP = 0
        IVAR = ' '
        DO WHILE (IVAR .NE. 'none' .AND. NIP .LT. 3)
          NIP = NIP + 1
          IVAR = 'none'
          VARPrompt = 'IVAR'//I4ToString(NIP)
          IVAR = GetCHARInput(VARPrompt,IVAR)
          IFoundit = .FALSE.
          DO 1010 I = 1,NParms
            IF (IVAR .EQ. PARMNAME(I)) THEN
              IPARM(NIP) = I
              IFoundit = .TRUE.
            END IF
1010      CONTINUE
          IF (.not. IFoundit .AND. IVAR .NE. 'none') THEN
            Message(1) = 'Variable not found: '//IVAR
            IOStat = ProcessIOString(Message,1,0,4)
            CommandBuffer = ' '
            NIP = NIP - 1
          END IF
          IF (IVAR .EQ. 'none') NIP = NIP - 1
        END DO

! This section shifts the IPARMS to the far right in the IPARM array.
! (That is if only 2 parameters are being swept, IPARM(3) becomes
! IPARM(2), IPARM(2) becomes IPARM(1) etc.)

        IF (NIP .EQ. 1) THEN
          IPARM(3) = IPARM(NIP)
        ELSE
          IF (NIP .EQ. 2) THEN
            IPARM(3) = IPARM(NIP)
            IPARM(2) = IPARM(NIP-1)
          END IF
        END IF
      GO TO 100
```

```
15    CONTINUE                ! Set the Do_TOTAL Flag
        Do_TOTAL = GetLOGICALInput('Do_TOTAL',Do_TOTAL)
      GO TO 100

20    CONTINUE                ! Set dependent variables to record
        NDP = 0
        DVAR = ' '
        DO WHILE (DVAR .NE. 'none' .AND. NDP .LT. 6)
          NDP = NDP + 1
          DVAR = 'none'
          VARPrompt = 'DVAR'//I4ToString(NDP)
          DVAR = GetCHARInput(VARPrompt,DVAR)
          IFoundit = .FALSE.
          DO 1020 I = 1,NCoeffs
            IF (DVAR .EQ. COEFFNAME(I)) THEN
              ICOEFF(NDP) = I
              IFoundit = .TRUE.
            END IF
1020        CONTINUE
          IF (.not. IFoundit .AND. DVAR .NE. 'none') THEN
            Message(1) = 'Variable not found: '//DVAR
            IOStat = ProcessIOString(Message,1,0,4)
            CommandBuffer = ' '
            NDP = NDP - 1
          END IF
          IF (DVAR .EQ. 'none') NDP = NDP - 1
        END DO
      GO TO 100

25    CONTINUE                ! Set a constant variable
        IVAR = GetCHARInput('IVAR',IVAR)
        IFoundit = .FALSE.
        DO 1015 I = 1,NParms
          IF (IVAR .EQ. PARMNAME(I)) THEN
            TEMPR4 = REAL(PARM(I))
            TEMPR4 = GetREALInput(IVAR,TEMPR4)
            PARM(I) = DBLE(TEMPR4)
            IFoundit = .TRUE.
          END IF
1015      CONTINUE
        IF (.not. IFoundit .AND. IVAR .NE. 'none') THEN
          Message(1) = 'Variable not found: '//IVAR
          IOStat = ProcessIOString(Message,1,0,4)
          CommandBuffer = ' '
          GO TO 25
        END IF
      GO TO 100

30    CONTINUE                ! Set Do_FLAPSCD logical
        Do_FLAPSCD = GetLOGICALInput('Do_FLAPSCD',Do_FLAPSCD)
```

```
      GO TO 100

40    CONTINUE                ! Set Do_COLLECTIVE logical
        Do_COLLECTIVE = GetLOGICALInput('Do_COLLECTIVE',Do_COLLECTIVE)
      GO TO 100

50    CONTINUE                ! Set the derrivative extraction flag
        DCDU = GetLOGICALInput('DCDU',DCDU)
      GO TO 100

55    CONTINUE                ! Set the Data sweep increments for IVARS
        DO 1025 I=3,4-NIP,-1
          J = INDEX(PARMNAME(IPARM(I)),' ') - 1
          VARPrompt12 = PARMNAME(IPARM(I))(:J)//'_inc'
          TEMPR4 = REAL(PARMINC(I))
          TEMPR4 = GetREALInput(VARPrompt12,TEMPR4)
          PARMINC(I) = DBLE(TEMPR4)
1025    CONTINUE
      GO TO 100

60    CONTINUE                ! Start the Data Sweep
        CALL SWEEPDATA(Do_FLAPSCD,Do_TOTAL,Do_COLLECTIVE,DCDU,NIP,
     .                 IPARM,NDP,ICOEFF)
      GO TO 100

70    CONTINUE                ! Quit the Data Sweep Utility
      GO TO 999

80    CONTINUE                ! Help requested, show Ivars and Dvars
        Message(1) = 'Available Independent Variables:'
        DO 1030 I = 1,NParms
          J = I + 1
          Message(J) = PARMNAME(I)//PARMDSCRP(I)
1030    CONTINUE
        Message(J+1) = ' '
        Message(J+2) = 'Available dependent Variables:'
        DO 1040 I = 1,Ncoeffs
          K = J + 2 + I
          Message(K) = COEFFNAME(I)//COEFFDSCRP(I)
1040    CONTINUE
        IOstat = ProcessIOString(Message,K,0,0)
      GO TO 100

85    CONTINUE               ! Reset all parameters to zero
        DO 1041 I = 1,NParms
          PARM(I) = 0.0
1041    CONTINUE
      GO TO 100

      END IF
! ---------------------------------------------------------------------
```

```
!
!     End Of SHELL_SWEEPDATA
!
! ---------------------------------------------------------------------
999   RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ---------------------------------------------------------------------
!
!       File Name: :INCLUDES:Cmd_Structure.txt
!
!  Command structure for simulation. Max. allowable commands set to 30.
!  Each menu uses its own custom number of commands. CmdH is the header
!  for each command menu. (More than one header can occurr in a menu)
!  CmdU and CmdL are commands in upper and lower case respectively. CmdD
!  is the command decription diplayed in the menu. HasSubMenu is a
!  logical flag for each command.
! ---------------------------------------------------------------------

      STRUCTURE / CMD_STRUC /
        INTEGER Num_Commands
        CHARACTER*40 CmdH(0:30)
        CHARACTER*4 CmdU(30)
        CHARACTER*4 CmdL(30)
        CHARACTER*40 CmdD(30)
        LOGICAL HasSubMenu(30)
      END STRUCTURE

C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ---------------------------------------------------------------------
!
!       File Name: :INCLUDES:SDCmd.txt
!
! ---------------------------------------------------------------------

      RECORD / CMD_STRUC / DSCMD

      IF (.NOT. Initialized) THEN

        DSCMD.Num_Commands = 11
        DSCMD.CmdH(0) = 'Data Sweep Utility: Commands'
        DSCMD.CmdH(1) = ' '
        DSCMD.CmdU(1) = 'COLL'
        DSCMD.CmdL(1) = 'coll'
        DSCMD.CmdD(1) = 'Set the Collective Controls Flag'
        DSCMD.HasSubMenu(1) = .FALSE.
        DSCMD.CmdH(2) = ' '
        DSCMD.CmdU(2) = 'TOT'
        DSCMD.CmdL(2) = 'tot'
```

```
  DSCMD.CmdD(2) = 'Set the Total Coefficient returned Flag'
  DSCMD.HasSubMenu(2) = .FALSE.
  DSCMD.CmdH(3) = ' '
  DSCMD.CmdU(3) = 'FLAP'
  DSCMD.CmdL(3) = 'flap'
  DSCMD.CmdD(3) = 'Set the Flap Scheduling Flag'
  DSCMD.HasSubMenu(3) = .FALSE.
  DSCMD.CmdH(4) = ' '
  DSCMD.CmdU(4) = 'DDU'
  DSCMD.CmdL(4) = 'ddu'
  DSCMD.CmdD(4) = 'Set the Derrivative extracting Flag'
  DSCMD.HasSubMenu(4) = .FALSE.
  DSCMD.CmdH(5) = ' '
  DSCMD.CmdU(5) = 'IVAR'
  DSCMD.CmdL(5) = 'ivar'
  DSCMD.CmdD(5) = 'Specify the Independent sweep variables'
  DSCMD.HasSubMenu(5) = .FALSE.
  DSCMD.CmdH(6) = ' '
  DSCMD.CmdU(6) = 'DVAR'
  DSCMD.CmdL(6) = 'dvar'
  DSCMD.CmdD(6) = 'Specify the Dependent sweep variables'
  DSCMD.HasSubMenu(6) = .FALSE.
  DSCMD.CmdH(7) = ' '
  DSCMD.CmdU(7) = 'SET'
  DSCMD.CmdL(7) = 'set'
  DSCMD.CmdD(7) = 'Set a Variable to a constant'
  DSCMD.HasSubMenu(7) = .FALSE.
  DSCMD.CmdH(8) = ' '
  DSCMD.CmdU(8) = 'INC'
  DSCMD.CmdL(8) = 'inc'
  DSCMD.CmdD(8) = 'Set IVAR Increments'
  DSCMD.HasSubMenu(8) = .FALSE.
  DSCMD.CmdH(9) = ' '
  DSCMD.CmdU(9) = 'RUN'
  DSCMD.CmdL(9) = 'run'
  DSCMD.CmdD(9) = 'Begin Data Sweep'
  DSCMD.HasSubMenu(9) = .FALSE.
  DSCMD.CmdH(10) = ' '
  DSCMD.CmdU(10) = 'HELP'
  DSCMD.CmdL(10) = 'help'
  DSCMD.CmdD(10) = 'List the available IVARS and DVARS'
  DSCMD.HasSubMenu(10) = .FALSE.
  DSCMD.CmdH(11) = ' '
  DSCMD.CmdU(11) = 'REST'
  DSCMD.CmdL(11) = 'rest'
  DSCMD.CmdD(11) = 'Reset all IVARS to zero'
  DSCMD.HasSubMenu(11) = .FALSE.

END IF
```

APPENDIX I.

1.2 The Sweep Data Algorithm

A. Usage

This module performs the data sweeps for as many as 3 independent parameters at a time, and saves the results as a Matlab binary workspace file.

A.1 SWEEPDATA

Function Prototype, SUBROUTINE

LOGICAL	DO_FLAPSCD, DO_TOTAL, DO_COLLECTIVE, DCDU
INTEGER	NIP, IPARM(3), NDP, ICOEFF(6)
COMMON	SWEEPPARMS, UEFFECTS

Assign data to all variables

CALL SWEEPDATA(DO_FLAPSCD, DO_TOTAL, DO_COLLECTIVE, DCDU, NIP, IPARM, NDP, ICOEFF)

SWEEPDATA increments the NIP parameters defined by IPARM() and saves the appropriate data to a specified Matlab workspace file.

Argument Definitions

DO_FLAPSCD	[in]	Enables/disables calls to flap scheduling functions.
DO_TOTAL	[in]	Total aero. coefficient data is returned if value is TRUE
DO_COLLECTIVE	[in]	Enables/disables the use of symmetric and differential control commands
DCDU	[in]	Enables/disables estimation of derivatives using a 4th order central difference approach.
NIP	[in]	Number of independent parameters to sweep.
IPARM()	[in]	Array indicating which parameters are to be swept
NDP	[in]	Number of dependent variables to save.
ICOEFF()	[in]	Indicates the dependent results that are to be saved

B. General Remarks

This module saves all of the results in the SWEEPPARMS and UEFFECTS common blocks (see section 1.1 for a description of these globals). It may also require some slight modifications in order to interface properly with different databases. This version is written for the F-15 ACTIVE database and should only be used as a template for other lookup routines. Part C below gives more details about the standard sections of this code.

C. Functional Description

When SWEEPDATA is called, it may need to initialize some database specific parameters. These should appear before any other executable statements. Next, the FILEEXIST parameter is set to FALSE (so that when the CREATE_MATFILE routine is called, it generates a new file), and the NCALLS array is set to a vector of 1's. The results are saved as double precision arrays that are sent to the Matlab External Interface libraries. The next section calculates the number of times that the database will have to be called based on the number of independent parameters and each parameter's increment (NCALLS). If any of these numbers exceed the size set for the data arrays, then an error message is generated and SWEEPDATA returns. Assuming these tests pass, the independent parameters are set

to their minimum allowable values, and a call is made to the shell interface routine GETFILENAME to request the name of the file to store the data. The database is then swept using a 3 level DO loop for the possible 3 independent sweep parameters. Within the inner most loop, (associated with the last independent parameter specified), the following tasks are performed:

1) The arguments for the table lookup routine (excluding control surfaces) should be defined. Recall that the independent parameters to sweep are stored in the PARM field of the SWEEPPARMS common block. So if PARM(3) were defined as angle of attack, then a statement should appear setting the angle of attack argument to PARM(3).
2) After all table lookup arguments have been defined, SWEEPDATA calls ADC2. This subroutine calculates static pressure PS, dynamic pressure QBAR, and true airspeed in feet/sec VT, based on the current Mach number XMACH, and altitude ALT.
3) If DO_FLAPSCD is enabled then the Flap Scheduling functions should be called to set the flap positions based on the current aircraft states. Note that if this feature is chosen not to be supported, either a dummy routine should be implemented, or the lines commented out so that link errors will not occur.
4) Next, the derivative extraction logic is performed. Derivative calculations are done by calling the table lookup routine within a loop. If DCDU is TRUE, the lower bound for the loop (LB) is -2 and the upper bound (UB) is 2, resulting in 5 calls to the lookup routine. For each call, the inner most parameter is given a small “delta” from its reference value (PTEMP), such that a 4th order central difference equation can be applied. If DCDU is FALSE then the upper and lower bounds are both set to 0 so that the database is only called once using the reference value for the 3rd independent parameter.
5) Within the database calling loop, the inner most independent parameter is given a small offset from the reference value if derivative extracting is enabled. The increment applied is based on the current loop count M and base “delta” value H. After this point, the table lookup arguments representing control surface deflections can be defined. and the database routine can be called.
6) If DO_TOTAL is TRUE, then the aerodynamic coefficients returned are defined to the MCOEFF array (this array is sent to the Matlab External Interface functions for export). Otherwise, the control specific intermediate results in the UEFFECTS common block are defined to the array. In addition, a temporary 6 by 5 array CTEMP stores the newly defined MCOEFF array for derivative extracting purposes. At this point, the lower bound/upper bound loop repeats as described in step 4.
7) If DCDU is enabled, then the MCOEFF array is redefined by a call to the EXTRACTD function using the entries of the CTEMP array. (This function returns the numerically calculated derivative). The MPARM array (this array is sent to the MEX libraries as well) is defined based on the number of independent sweep parameters, and the inner most parameter is incremented. This ends the inner most loop

For each middle parameter value, (the next to last independent parameter specified), the procedures above are performed for the entire allowable range of the inner most parameter. When this is completed, the inner parameter is reset to its minimum, the 2nd parameter is incremented, and the loop begins again. When the 2nd parameter has been swept through its allowable range, the 6 K x J MCOEFF matrices contain data associated with the Kth inner most parameter and the Jth middle parameter. In other words, the rows of the matrix contain data variations associated with the middle parameter changing and the inner most parameter constant, while the matrix columns contain data variations due to the inner most parameter, holding the middle parameter constant. The requested matrices are exported to a Matlab file using a call to the shell interface routine CREATE_MATFILE (This routine in turn references the Matlab External Interface libraries). If three parameters are being swept, then each matrix name is appended with a number representing the current outer-most parameter index. For example, assume a Mach number (MACH), angle of attack (ALPHA) and left stabilator (DSL) sweep is being performed, and the Drag coefficient effects (CD) are being recorded. If the MACH number at the end of the middle parameter loop is the 3rd value between its minimum and maximum allowable range, then the exported matrix will be given the name CDALPHADSL3.

After the matrix data is exported and if a 3 level sweep is being performed, the inner most and middle parameters are reset to the minimum values, the outer-most parameter is incremented, and the loop continues, using a FILEEXIST value of TRUE so that the additional data is appended to the specified file. If only two parameters are being swept or the 3 level sweep is finished, then the independent parameter vectors are saved to the Matlab file, and SWEEPDATA returns.

D. Errors and Restrictions

Current restrictions on the number of available independent parameters, dependent parameters, and number of database calls are 29, 6, and 240 respectively. These can be changed by modifying the PARAMETER statements, and recompiling the executable.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: SWEEPDATA
!           Called By: SHELL_DATASWEEP
!           Calls to: AeroDBLibrary, MATLAB external interface library.
!                    (Consult The Matlab MEX technical documentation.)
!
! ------------------------------------------------------------------------
      SUBROUTINE SWEEPDATA (Do_FLAPSCD,Do_TOTAL,Do_COLLECTIVE,DCDU,NIP,
     .                      IPARM,NDP,ICOEFF)
! ------------------------------------------------------------------------
!
!       Function:   Varies specified independent parameter(s) and records
!                  the specified dependent parameter(s). Data is exported
!                  in MATLAB binary format (.MAT.)
!
! ------------------------------------------------------------------------
!
!     Glossary Section
!                                Locals
! Variable      Type                    Description
*Do_FLAPSCD    LOGICAL      Determines if Flaps are scheduled or not.
*Do_TOTAL      LOGICAL      Determines if total coefficients are returned
C                           or individual increments due to controls.
*Do_COLLECTIVE LOGICAL      Determines if Collective symmetric and diff
C                           controls are used instead of Left and Right
*DCDU          LOGICAL      ∂(Coefficient)/∂(Control) flag. F - do not
C                           calculate derrivatives, T - Calculate der-
C                           rivatives using a 5 point formula.
*NIP           INTEGER      number of independent parameters to sweep
*IPARM         INTEGER()    array indicating which variables are to be
C                           swept.
*NDP           INTEGER      number of coefficents to record
*ICOEFF        INTEGER()    array indicating which coefficients are to
C                           be returned.
```

```
!
! ---------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                          By
!  DEC 04 1995        Created                                J. Bolling
!  JUN 05 1996        Began making extensive modifications to work
!                     with the F-15 SDF database and the V1.5 Shell
!                     interface.                                    J.B.
!  JUN 14 1996        Added the Derrivative extracting logic.       J.B.
!
! ---------------------------------------------------------------------
      IMPLICIT NONE
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
! -----------------------------Inputs----------------------------------
      INTEGER NIP, NDP, IPARM(3), ICOEFF(3)
      LOGICAL Do_FLAPSCD, Do_TOTAL, Do_COLLECTIVE, DCDU
! ---------------------------Sweep Parameters--------------------------
      INTEGER NParms,NCoeffs
      PARAMETER (NParms = 29, NCoeffs = 6)
      DOUBLE PRECISION PARM(NParms), PARMMIN(NParms), PARMMAX(NParms),
     .                 COEFF(NParms,NCoeffs), UEFF(NParms,NCoeffs),
     .                 PARMINC(3)
      CHARACTER*8 PARMNAME(NParms), COEFFNAME(NCoeffs),
     .            UEFFNAME(NParms,NCoeffs)
      CHARACTER*32 PARMDSCRP(NParms), COEFFDSCRP(NCoeffs)
! -------------------------Surface Commsnds----------------------------
      DOUBLE PRECISION  SURCOM(41)
      DOUBLE PRECISION
     .   DAILD,     DAILL_L,    DAILR_L,    DCANRL,    DCANRR,
     .   DCANRD,    DDCAND,     DDNOZD,     DDTEF,     DNOZD,
     .   DNOZL,     DNOZR,      DRAILD,     DROTVB,    DROTVD,
     .   DROTVT,    DRUDD,      DRUDL,      DRUDR,     DSPLD,
     .   DSTBD,     DSTBL_L,    DSTBR_L,    DTALD,     FLAPL,
     .   FLAPR,     NOZY,       NOZP,       NDUM03,    NDUM04,
     .   NDUM05,    NDUM06,     NDUM07,     NDUM08,    NDUM09,
     .   NDUM10,    NDUM11,     NDUM12,     NDUM13,    NDUM14,
     .   NDUM15
! -----------------------------Locals----------------------------------
      REAL ALFA, BETA, XMACH, P, Q, R, ALT, VT, QBAR, PS, CLLT, CMT,
     .     CNT , CLT , CDT  , CYT
      INTEGER I,J,K,JJ,JJJ,KK,NCALLS(3),Matrows,Matcolumns,L,LL,
     .        CINDEX1,CINDEX2,CINDEX3,M,LB,UB
      INTEGER Maxarraysize,Maxarraysize2
      PARAMETER (Maxarraysize = 240)
      PARAMETER (Maxarraysize2 = Maxarraysize**2)
      CHARACTER*16 Matname
```

```
      CHARACTER*20 Filename, Header
      CHARACTER*16 Filetype
      CHARACTER*4 File_ext
      PARAMETER (File_ext = '.mat')
      REAL*8 MCOEFF(6,Maxarraysize,Maxarraysize)
      REAL*8 MPARM(Maxarraysize,3),Matdata(Maxarraysize2)
      LOGICAL Fileexist,CallEntry
      REAL NPRNR4,LEXA,REXA,H
      PARAMETER (H = 0.01)
      DOUBLE PRECISION CSMODE
      DOUBLE PRECISION CTEMP(6,-2:2),PTEMP
! ----------------------------Functions---------------------------------
      CHARACTER*8 I4toString
      DOUBLE PRECISION EXTRACTD
! ----------------------------------------------------------------------
!
!     Common Section
!
! ----------------------------------------------------------------------
      COMMON / SWEEPPARMS / PARM, PARMNAME, PARMDSCRP, PARMMIN, PARMMAX,
     .                      PARMINC, COEFF, COEFFNAME, COEFFDSCRP
      COMMON / UEFFECTS / UEFF, UEFFNAME
      COMMON / SURFACE /  SURCOM
! ----------------------------------------------------------------------
!
!     Eauivalence Section
!
! ----------------------------------------------------------------------
C*  EQUIVALENCE STATEMENTS FOR SURFACE COMMANDS

      EQUIVALENCE (SURCOM( 1),DAILD),   (SURCOM( 2),DAILL_L)
      EQUIVALENCE (SURCOM( 3),DAILR_L), (SURCOM( 4),DCANRL)
      EQUIVALENCE (SURCOM( 5),DCANRR),  (SURCOM( 6),DCANRD)
      EQUIVALENCE (SURCOM( 7),DDCAND),  (SURCOM( 8),DDNOZD)
      EQUIVALENCE (SURCOM( 9),DDTEF),   (SURCOM(10),DNOZD)
      EQUIVALENCE (SURCOM(11),DNOZL),   (SURCOM(12),DNOZR)
      EQUIVALENCE (SURCOM(13),DRAILD),  (SURCOM(14),DROTVB)
      EQUIVALENCE (SURCOM(15),DROTVD),  (SURCOM(16),DROTVT)
      EQUIVALENCE (SURCOM(17),DRUDD),   (SURCOM(18),DRUDL)
      EQUIVALENCE (SURCOM(19),DRUDR),   (SURCOM(20),DSPLD)
      EQUIVALENCE (SURCOM(21),DSTBD),   (SURCOM(22),DSTBL_L)
      EQUIVALENCE (SURCOM(23),DSTBR_L), (SURCOM(24),DTALD)
      EQUIVALENCE (SURCOM(25),FLAPL),   (SURCOM(26),FLAPR)
      EQUIVALENCE (SURCOM(27),NOZY),    (SURCOM(28),NOZP)
      EQUIVALENCE (SURCOM(29),NDUM03),  (SURCOM(30),NDUM04)
      EQUIVALENCE (SURCOM(31),NDUM05),  (SURCOM(32),NDUM06)
      EQUIVALENCE (SURCOM(33),NDUM07),  (SURCOM(34),NDUM08)
      EQUIVALENCE (SURCOM(35),NDUM09),  (SURCOM(36),NDUM10)
      EQUIVALENCE (SURCOM(37),NDUM11),  (SURCOM(38),NDUM12)
      EQUIVALENCE (SURCOM(39),NDUM13),  (SURCOM(40),NDUM14)
      EQUIVALENCE (SURCOM(41),NDUM15)
```

```
! ------------------------------------------------------------------
!
!     Initialization Section
!
! ------------------------------------------------------------------
      DATA CSMODE / 1.0 /
      DATA LEXA,REXA /6.6, 6.6/
      Fileexist = .FALSE.
      NCALLS(1) = 1
      NCALLS(2) = 1
      NCALLS(3) = 1

! find how many calls we will have to make for the data sweep.

      DO 30 I = 3,4-NIP,-1

        NCALLS(I) = NINT((PARMMAX(IPARM(I)) - PARMMIN(IPARM(I)))/
     .               PARMINC(I)) + 1
        IF (NCALLS(I) .GT. Maxarraysize) THEN
          WRITE(*,*) 'ERROR: Maximum array size exceeded in SWEEPDATA'
          RETURN
        END IF
! Set the IVARS that we are sweeping to their minimums

        PARM(IPARM(I)) = PARMmin(IPARM(I))

30    CONTINUE

! Get the name of the file to store data to (Shell Interface call)

      Header = 'Data Sweep'
      CALL GetFilename (Header,File_ext,Filename,Filetype)

! ------------------------------------------------------------------
!
!     Run Section
!
! ------------------------------------------------------------------
! Sweep database

      DO 40 I = 1,NCALLS(1)
        WRITE(*,*) 'GETTING DATA...'
        DO 50 J = 1,NCALLS(2)
          DO 60 K = 1,NCALLS(3)
            ALFA  = REAL(PARM(1))
            BETA  = REAL(PARM(3))
            XMACH = REAL(PARM(2))
            P     = REAL(PARM(4))
            Q     = REAL(PARM(5))
            R     = REAL(PARM(6))
            ALT   = REAL(PARM(7))
```

```
            NPRNR4 = MAX(REAL(PARM(8)),1.0)
            CALL ADC2(XMACH, ALT, VT, QBAR, PS)
! --------------------Schedule the flaps if desired--------------------
          IF (Do_FLAPSCD) THEN
!            CALL FLAPSCD
          END IF
! ----------Supporting Logic for derrivative extracting---------------
          IF (DCDU) THEN
            LB = -2
            UB = 2
            PTEMP = PARM(IPARM(3))
          ELSE
            LB = 0
            UB = 0
            PTEMP = PARM(IPARM(3))
          END IF
! --------------------Call the database here--------------------------
      DO 61 M = LB,UB,1

        IF (DCDU) THEN
          PARM(IPARM(3)) = PTEMP + M*H                ! Used for DCDU
        END IF

        DCANRL  = PARM( 9)
        DCANRR  = PARM(10)
        DAILL_L = PARM(11)
        DAILR_L = PARM(12)
        FLAPL   = PARM(13)
        FLAPR   = PARM(14)
        DRUDD   = PARM(15)
        DRUDD   = PARM(16)
        DSTBL_L = PARM(17)
        DSTBR_L = PARM(18)
        NOZP    = PARM(19)
        NOZY    = PARM(20)
        DCANRD  = PARM(21)
        DDCAND  = PARM(22)
        DRAILD  = PARM(23)
        DAILD   = PARM(24)
        NDUM03  = PARM(25)
        DDTEF   = PARM(26)
        DSTBD   = PARM(27)
        DTALD   = PARM(28)
        DRUDD   = PARM(29)

        CALL F15AERO(CDT, CLT, CMT, CYT, CNT, CLLT, ALFA, XMACH, QBAR,
     .               VT,  P,   Q,   R,   ALT, 0.0,  BETA, 0.0,   0.0,
     .               0.0, NPRNR4, LEXA, REXA, CSMODE, .TRUE.,
     .               Do_COLLECTIVE)
```

```
! ---------------------------------------------------------------------

          IF (Do_TOTAL) THEN
            MCOEFF(1,K,J) = DBLE(CDT)
            MCOEFF(2,K,J) = DBLE(CLT)
            MCOEFF(3,K,J) = DBLE(CMT)
            MCOEFF(4,K,J) = DBLE(CYT)
            MCOEFF(5,K,J) = DBLE(CLLT)
            MCOEFF(6,K,J) = DBLE(CNT)
          ELSE
            MCOEFF(1,K,J) = UEFF(IPARM(3),1)
            MCOEFF(2,K,J) = UEFF(IPARM(3),2)
            MCOEFF(3,K,J) = UEFF(IPARM(3),3)
            MCOEFF(4,K,J) = UEFF(IPARM(3),4)
            MCOEFF(5,K,J) = UEFF(IPARM(3),5)
            MCOEFF(6,K,J) = UEFF(IPARM(3),6)
          END IF

! more DCDU stuff
            CTEMP(1,M) = MCOEFF(1,K,J)
            CTEMP(2,M) = MCOEFF(2,K,J)
            CTEMP(3,M) = MCOEFF(3,K,J)
            CTEMP(4,M) = MCOEFF(4,K,J)
            CTEMP(5,M) = MCOEFF(5,K,J)
            CTEMP(6,M) = MCOEFF(6,K,J)

61        CONTINUE

! Extract the derrivatives here for each dependent variable

          IF (DCDU) THEN
            DO 62 M = 1,6
              MCOEFF(M,K,J) = EXTRACTD(CTEMP(M,-2),CTEMP(M,-1),
    .                                  CTEMP(M, 1),CTEMP(M, 2),H)
62          CONTINUE
            PARM(IPARM(3)) = PTEMP              ! Retore to original val.
          END IF

          MPARM(K,3) = PARM(IPARM(3))
          IF (NIP .GT. 1) MPARM(J,2) = PARM(IPARM(2))
          IF (NIP .GT. 2) MPARM(I,1) = PARM(IPARM(1))
          PARM(IPARM(3)) = PARM(IPARM(3)) + PARMINC(3)

60      CONTINUE

        PARM(IPARM(3)) = PARMMIN(IPARM(3))
        IF (NIP .GT. 1) THEN
          PARM(IPARM(2)) = PARM(IPARM(2)) + PARMINC(2)
        ENDIF

50    CONTINUE
```

```
! ---------------------------------------------------------------------
! Export the data that we have so far (This is the matrix of coefficient
! data. for the IPARM2 and IPARM3 sweeps. There will will be more matri-
! ces to follow for each consecutive IPARM1
! ---------------------------------------------------------------------
      WRITE(*,*) 'EXPORTING DATA...'

      DO 75 JJ=1,NDP

        IF (NIP .EQ. 1) THEN
          IF (Do_TOTAL) THEN
            CINDEX1 = INDEX(COEFFNAME(ICOEFF(JJ)),' ') - 1
            Matname = COEFFname(ICOEFF(JJ))(:CINDEX1)//
     .                PARMname(IPARM(3))
          ELSE
            CINDEX1 = INDEX(UEFFNAME(IPARM(3),ICOEFF(JJ)),' ') - 1
            Matname = UEFFNAME(IPARM(3),ICOEFF(JJ))(:CINDEX1)
          ENDIF
          Matrows = NCALLS(3)
          Matcolumns = 1
        ENDIF

        IF (NIP .EQ. 2) THEN
          IF (Do_TOTAL) THEN
            CINDEX3 = INDEX(PARMNAME(IPARM(3)),' ') - 1
            CINDEX2 = INDEX(PARMNAME(IPARM(2)),' ') - 1
            CINDEX1 = INDEX(COEFFNAME(ICOEFF(JJ)),' ') - 1
            Matname = COEFFname(ICOEFF(JJ))(:CINDEX1)//
     .                PARMname(IPARM(2))(:CINDEX2)//
     .                PARMname(IPARM(3))(:CINDEX3)
          ELSE
            CINDEX3 = INDEX(UEFFNAME(IPARM(3),ICOEFF(JJ)),' ') - 1
            Matname = UEFFNAME(IPARM(3),ICOEFF(JJ))(:CINDEX3)
          END IF
          Matrows = NCALLS(3)
          Matcolumns = NCALLS(2)
        END IF

        IF (NIP .EQ. 3) THEN
          IF (Do_TOTAL) THEN
            CINDEX3 = INDEX(PARMname(IPARM(3)),' ') - 1
            CINDEX2 = INDEX(PARMname(IPARM(2)),' ') - 1
            CINDEX1 = INDEX(COEFFNAME(ICOEFF(JJ)),' ') - 1
            Matname = COEFFname(ICOEFF(JJ))(:CINDEX1)//
     .                PARMname(IPARM(2))(:CINDEX2)//
     .                PARMname(IPARM(3))(:CINDEX3)//
     .                I4toString(I)
          ELSE
            CINDEX3 = INDEX(UEFFNAME(IPARM(3),ICOEFF(JJ)),' ') - 1
            Matname = UEFFNAME(IPARM(3),ICOEFF(JJ))(:CINDEX3)//
```

```
     .                I4toString(I)
          END IF
          Matrows = NCALLS(3)
          Matcolumns = NCALLS(2)
        ENDIF

        L = 0
        WRITE(*,'(15x,A)') Matname
        DO 80 JJJ = 1,NCALLS(2)
          DO 85 KK = 1,NCALLS(3)
           Matdata(L+KK) = MCOEFF(ICOEFF(JJ),KK,JJJ)
           LL = KK
85        CONTINUE
          L = L + LL
80      CONTINUE
        CALL Create_MATFile(Filename,Fileexist,Matname,Matrows,
     .                      Matcolumns,Matdata)
        Fileexist = .TRUE.
75    CONTINUE

! ---------------------------------------------------------------------
! Finished exporting the matrices, continue with incrementing IPARM1 or
! finsh by exporting the rest of the variables.
! ---------------------------------------------------------------------

        PARM(IPARM(3)) = PARMMIN(IPARM(3))
        IF (NIP .GT. 1) PARM(IPARM(2)) = PARMMIN(IPARM(2))
        IF (NIP .GT. 2) THEN
          PARM(IPARM(1)) = PARM(IPARM(1)) + PARMINC(1)
        ENDIF

40    CONTINUE

! ---------------------------------------------------------------------
! End of Sweep
!
! Finish ".MAT" File export
! ---------------------------------------------------------------------

      Matname = PARMname(IPARM(3))
      Matrows = NCALLS(3)
      Matcolumns = 1
      WRITE(*,'(15x,A)') Matname
      DO 65 J=1,NCALLS(3)
        Matdata(J) = MPARM(J,3)
65    CONTINUE
      CALL Create_MATFile(Filename,Fileexist,Matname,Matrows,
     .                    Matcolumns,Matdata)
      IF (NIP .GT. 1) THEN
        Matname = PARMname(IPARM(2))
        Matrows = NCALLS(2)
```

```
        Matcolumns = 1
        WRITE(*,'(15x,A)') Matname
        DO 70 J=1,NCALLS(2)
          Matdata(J) = MPARM(J,2)
70      CONTINUE
        CALL Create_MATFile(Filename,Fileexist,Matname,Matrows,
     .                      Matcolumns,Matdata)
      ENDIF

      IF (NIP .GT. 2) THEN
        Matname = PARMname(IPARM(1))
        Matrows = NCALLS(1)
        Matcolumns = 1
        WRITE(*,'(15x,A)') Matname
        DO 1200 J=1,NCALLS(1)
          Matdata(J) = MPARM(J,1)
1200    CONTINUE
        CALL Create_MATFile(Filename,Fileexist,Matname,Matrows,
     .                      Matcolumns,Matdata)
      ENDIF
!
! End of MAT File export
!
! ----------------------------------------------------------------------
!
!       END OF SWEEPDATA
!
! ----------------------------------------------------------------------
999   RETURN
      END
```

1.3 Miscellaneous Functions and Subroutines

A. Usage

These small functions and subroutines perform more general calculations not specific to the SWEEPDATA module but are required for execution.

A.1 ADC2

Function Prototype, SUBROUTINE

REAL	MACH, ALT, VT, QBAR, PS

Assign values to MACH and ALT

```
CALL ADC2(MACH, ALT, VT, QBAR, PS)
```

ADC2 calculates the values of VT, QBAR, and PS

Argument Definitions

MACH	[in]	Mach number (ND)
ALT	[in]	Altitude (ft)
VT	[out]	True airspeed (ft/sec)
QBAR	[out]	Dynamic pressure (psf)
PS	[out]	Static pressure (psf)

A.2 EXTRACTD

Function Prototype, REAL*8

REAL*8	V1, V2, V3, V4, H

Assign values to all arguments

```
D = EXTRACTD(V1, V2, V3, V4, H)
```

EXTRACTD returns a 4th order central difference derivative approximation based on the values V1 - V4 and step size H.

Argument Definitions

V1	[in]	reverence value - 2*H
V2	[in]	reverence value - 1*H
V3	[in]	reverence value + 1*H
V4	[in]	reverence value + 2*H
H	[in]	Step size

B. General Remarks

ADC2 is a modified form of the standard atmosphere model (subroutine ADC) found in appendix A of *Aircraft Control and Simulation* (See reference 16). The difference is that Mach number is treated as a known value to calculate velocity.

EXTRACTD uses a simple 5 point formula to approximate the derivative

C. Source Listing

```
**********************************************************************
        SUBROUTINE ADC2(AMACH, ALT, VT, QBAR, PS)
        REAL VT, ALT, AMACH, QBAR
        REAL R0,TFAC,T,RHO,PS
        DATA R0/2.377E-3/
        TFAC = 1.0 - 0.703E-5 * ALT
        T = 519.0*TFAC
        IF (ALT.GE.35000.0) T=390.0
        RHO = R0*(TFAC**4.14)
        VT = AMACH*SQRT(1.4*1716.3*T)
        QBAR=0.5*RHO*VT*VT
        PS=1715.0*RHO*T
        RETURN
        END
C **********************************************************************
      FUNCTION EXTRACTD(V1,V2,V3,V4,H)
      IMPLICIT NONE
      DOUBLE PRECISION EXTRACTD, V1, V2, V3, V4
      REAL H
      EXTRACTD = (V1 - 8.0*V2 + 8.0*V3 - V4)/(12.0*H)
      RETURN
      END
```

2. Creating Mesh Constants for 2D and 3D Tables

A. Usage

The functions MMCS2D (Make Mesh Constants) and MMCS3D can be used to generate the affine table lookup mesh constants for each table block

A.1 MMCS2D

Function Prototype, Matlab m-file

This script file has no arguments. At the Matlab prompt:

»MMCS2D

MMCS2D prompts the user for the required information, calculates the mesh constants, and saves the data to the generic MCSData.mat file

Argument Definitions

none

A.2 MMCS3D

Function Prototype, Matlab m-file

This script file has no arguments. At the Matlab prompt:

»MMCS3D

MMCS3D prompts the user for the required information, calculates the mesh constants, and saves the data to the generic MCSData.mat file

Argument Definitions

none

B. General Remarks

The MMCS functions require the following input from the user:

1) the name of the Matlab file that contains the Sweep Data results
2) the function (or function family name) for which to generate the mesh constants.
3) the independent parameters.

Make sure that when entering the independent parameters, they are entered in the correct order. For reference, Sweep Data stores the dependent variations in columns. These columns are stacked side by side for two dimensional data sweeps, and for three dimensional sweeps, slices associated with the outer-most independent parameter are stored as separate matrices.

A Note about the MCSData.mat file:

After the mesh constants have been calculated for the complete table range, the MCSData.mat file is created. This file contains the “C_” mesh constants, the minimum values for each independent parameter (P_min), and the independent parameter increments (P_inc). The underscore for each mesh constant name is replaced by the parameters that is multiplies. For instance, CP1P3 represents the mesh constant that multiplies the 1st (outer-most parameter) and the 3rd (inner most) parameters. The underscores in the P_min and P_inc variables are replaced by their respective parameter number (ie. P1min and P1inc).

C. Functional Description

MMCS2D first clears the workspace and requests the name of the file to load. It then displays all of the variables saved in the file so that the user will easily be able to enter the correct parameters. MMCS2D prompts for the function matrix to be used to generate the mesh constants, followed by the independent parameters associated with the rows and columns of the matrix. Using this data, the number of table blocks are found (P2meshsize X P1meshsize), the minimum and maximum values for the independent parameters are extracted, and the parameter increments are calculated. After the mesh constant matrices are initialized to zeros, a 2 level DO loop is performed to calculate the 4 mesh constants CP1, CP2, CP1P2, and C0 for each block in the data table by solving a system of 4 equations for the 4 known node-defining points. After all of the mesh constants have been calculated, the data is saved to the MCSData.mat file, the workspace is cleared, and the script ends.

MMCS3D performs the same procedures as MMCS2D, modified to account for a 3rd dimension. Basically, this adds a 3rd DO loop, and necessitates the calculations of 8 mesh constants CP1P2P3, CP1P2, CP2P3, CP1P3, CP1, CP2, CP3, and C0. The user input section however, is slightly different. MMCS3D asks for the name of the file to load and displays its variables as usual, and proceeds with a function name request. The request here is different from that in MMCS2D in that MMCS3D only needs the function “family” name. Thus, the index numbers appended to the end of each matrix should be omitted. Next, the independent parameters associated with the different matrices, matrix rows, and matrix columns are requested.

D. Errors and Restrictions

Matlab requires that character data be enclosed in quotes. Therefore, the function family name in MMCS3D and the requested filenames to load in MMCS3D and MMCS2D must be typed within quotes or an error will occur.

E. Source Listing

```
% MMCS2D (Make Mesh Constants; 2-D Tables)
%     This script generates the 4 constants for each block in the 2
%     dimensional Affine interpolation tables.

   clear;

% Load a file containing the interpolation data

   eval(['load ',(input('Filename: '))]);
   whos
   func = input('Function to generate mesh constants> ');

% set up the independent variable parameters
```

```
   parm1     = input('1st Independent parameter (Row Data)> ');
   parm2     = input('2nd Indepentent parameter (Column Data)> ');
   P1meshsize = length(parm1) - 1;
   P2meshsize = length(parm2) - 1;
   P1min = parm1(1);
   P1max = parm1(length(parm1));
   P2min = parm2(1);
   P2max = parm2(length(parm2));
   P1inc = (P1max - P1min)/P1meshsize;
   P2inc = (P2max - P2min)/P2meshsize;

% initialize the coefficient matrices

   CP1   = zeros(P2meshsize,P1meshsize);
   CP1P2 = zeros(P2meshsize,P1meshsize);
   CP2   = zeros(P2meshsize,P1meshsize);
   Co    = zeros(P2meshsize,P1meshsize);

% Start generating mesh constants

   X0 = P1min;
   X1 = P1min + P1inc;
   D0 = P2min;
   D1 = P2min + P2inc;
   for i = 1:P1meshsize;
     for j = 1:P2meshsize;
      F = [func(j,i),func(j+1,i),func(j,i+1),func(j+1,i+1)]';
      A = [X0,D0,X0*D0,1;X0,D1,X0*D1,1;X1,D0,X1*D0,1;X1,D1,X1*D1,1];
      C = inv(A)*F;
      CP1(j,i)   = C(1);
      CP2(j,i)   = C(2);
      CP1P2(j,i) = C(3);
      Co(j,i)    = C(4);
      D0 = D1;
      D1 = D1 + P2inc;
     end;
% get ready for next mesh
     D0 = P2min;
     D1 = P2min + P2inc;
     X0 = X1;
     X1 = X1 + P1inc;
   end;

% Export the data to a file

   save MCSData CP1 CP1P2 CP2 Co P1min P2min P1inc P2inc;
   clear;
   end;


% MMCS3D (Make Mesh Constants; 3-D Tables)
```

```
%  This script generates the 8 constants for each block in the 3
%     dimensional Affine interpolation tables.
%
   clear;

% Load a file containing the interpolation data

   eval(['load ',(input('Filename: '))]);
   whos
   funname = input('Enter the Function Name>');

% set up the independent variable parameters

   parm1 = input('1st Independent parameter (Matrix Data)>');
   parm2 = input('2nd Indepentent parameter (Row Data)>');
   parm3 = input('3rd Independent parameter (Column Data)>');
   P1meshsize = length(parm1) - 1;
   P2meshsize = length(parm2) - 1;
   P3meshsize = length(parm3) - 1;
   P1min = parm1(1);
   P1max = parm1(length(parm1));
   P2min = parm2(1);
   P2max = parm2(length(parm2));
   P3min = parm3(1);
   P3max = parm3(length(parm3));
   P1inc = (P1max - P1min)/P1meshsize;
   P2inc = (P2max - P2min)/P2meshsize;
   P3inc = (P3max - P3min)/P3meshsize;

% Initialize the Coefficients

   CP1      = zeros(P1meshsize*P3meshsize,P2meshsize);
   CP2      = zeros(P1meshsize*P3meshsize,P2meshsize);
   CP3      = zeros(P1meshsize*P3meshsize,P2meshsize);
   CP1P2 = zeros(P1meshsize*P3meshsize,P2meshsize);
   CP1P3 = zeros(P1meshsize*P3meshsize,P2meshsize);
   CP2P3 = zeros(P1meshsize*P3meshsize,P2meshsize);
   CP1P2P3  = zeros(P1meshsize*P3meshsize,P2meshsize);
   Co    = zeros(P1meshsize*P3meshsize,P2meshsize);

% Start generating the mesh constants (X <-> Parm1;Y <-> Parm2;Z <-> Parm3)

   X0 = P1min;
   X1 = P1min + P1inc;
   Y0 = P2min;
   Y1 = P2min + P2inc;
   Z0 = P3min;
   Z1 = P3min + P3inc;

   for i = 1:P1meshsize;
      matname = [funname,int2str(i)];
```

```
      eval(['func1 = [',matname,'];']);
      matname = [funname,int2str(i+1)];
      eval(['func2 = [',matname,'];']);
      for j = 1:P2meshsize;
         for k = 1:P3meshsize;
            k0 = (i-1)*P3meshsize + k;
            F = [func1(k,j),func1(k,j+1),func1(k+1,j),func1(k+1,j+1),...
                  func2(k,j),func2(k,j+1),func2(k+1,j),func2(k+1,j+1)]';
            A = [X0*Y0*Z0,X0*Y0,X0*Z0,Y0*Z0,X0,Y0,Z0,1;...
                  X0*Y1*Z0,X0*Y1,X0*Z0,Y1*Z0,X0,Y1,Z0,1;...
                  X0*Y0*Z1,X0*Y0,X0*Z1,Y0*Z1,X0,Y0,Z1,1;...
                  X0*Y1*Z1,X0*Y1,X0*Z1,Y1*Z1,X0,Y1,Z1,1;...
                  X1*Y0*Z0,X1*Y0,X1*Z0,Y0*Z0,X1,Y0,Z0,1;...
                  X1*Y1*Z0,X1*Y1,X1*Z0,Y1*Z0,X1,Y1,Z0,1;...
                  X1*Y0*Z1,X1*Y0,X1*Z1,Y0*Z1,X1,Y0,Z1,1;...
                  X1*Y1*Z1,X1*Y1,X1*Z1,Y1*Z1,X1,Y1,Z1,1];
            C = inv(A)*F;
            CP1P2P3(k0,j)     = C(1);
            CP1P2(k0,j)       = C(2);
            CP1P3(k0,j)       = C(3);
            CP2P3(k0,j)       = C(4);
            CP1(k0,j)         = C(5);
            CP2(k0,j)         = C(6);
            CP3(k0,j)         = C(7);
            Co(k0,j)          = C(8);

            Z0 = Z1;
            Z1 = Z1 + P3inc;
         end;
         Z0 = P3min;
         Z1 = Z0 + P3inc;
         Y0 = Y1;
         Y1 = Y1 + P2inc;
      end;
      Z0 = P3min;
      Z1 = Z0 + P3inc;
      Y0 = P2min;
      Y1 = Y0 + P2inc;
      X0 = X1;
      X1 = X1 + P1inc;
   end;

% Export the data to a file

   save MCSData CP1P2P3 CP1P2 CP1P3 CP2P3 CP1 CP2 CP3 Co P1min P2min P3min...
             P1inc P2inc P3inc;
   clear;
   end;
```

APPENDIX I.

APPENDIX I.

3. Converting the MCS Output Files to ASCII Files

A. Usage

The MAT2ASCII function converts the mesh constant data in the MCSData.mat file created by one of the MMCS scripts into an ASCII text file that can be read by the Control Allocation INITUEFF module.

A.1 MAT2ASCII

Function Prototype, Matlab m-file

This script file has no arguments. At the Matlab prompt:

»mat2ascii

MAT2ASCII will convert a binary (*.mat) file into an ASCII text formatted file

Argument Definitions

none

B. General Remarks

MAT2ASCII conforms to the FORTRAN77 standard by not writing any data past the 72nd column. This often makes the mesh constant matrices hard for the human eye to read since it is not uncommon to have to continue each matrix row across multiple lines. This feature can be overridden by setting the ISF77 flag to 0.

C. Functional Description

MAT2ASCII begins by clearing the workspace and setting the ISF77 flag appropriately (see B. above). It then requests the name of the MCSData file to load, the name of the ASCII file to create (asciiname), and the number of table dimensions for the data represented in the file. The size of the mesh constant matrices is calculated and stored in matsize. The format to write the matrix rows in is determined in a DO loop. If ISF77 is set to 1, then MAT2ASCII writes at most 4 E14.6 numbers per text line. The locations of the carriage returns in the format string depend on 1.) whether or not the 4 numbers per line constraint is violated and 2.) whether or not the format string accounts for every entry in the matrix row. Once the format string is created, the data (with comments) is exported to the file specified by asciiname.

D. Errors and Restrictions

Matlab requires that character data be enclosed in quotes. Note this limitation when specifying filenames.

E. Source Listing

```
% mat2ascii: This script writes all of the Affine Interpolation constants
%           to a file in ascii format. It adheres to the FORTRAN 77 standard
%           of not writting past the 72nd column. To write a regular table
%           without conforming to FORTRAN 77's needs, set ISF77 to 0.
%
      clear;
```

```
% ISF77 = 1 -> Conform to the Fortran77 standard.
% ISF77 = 0 -> Do not conform to the Fortran77 standard.

      ISF77 = 1;

% Load the data file

      eval (['load ',input('Enter a .mat file to load>')]);
      asciiname = input('Save ascii data to: ');
      tbldim = input('Table Dimensions: ');

% start creating the format string

      matsize = size(Co);
      formatstr = [' %+14.6e'];
      j = 1;

      for i = 2:matsize(2);
            j = j + ISF77;
            if (j < 4 & i < matsize(2))
                  formatstr = [formatstr,' %+14.6e'];
            end
            if (j == 4)
                  formatstr = [formatstr,' %+14.6e\n'];
                  j = 0;
            end
            if (i == matsize(2))
                  formatstr = [formatstr,' %+14.6e\n'];
            end
      end;

% output data

      if (tbldim == 2)
            fid = fopen(asciiname,'w');
            fprintf(fid,'c \nc CP1P2\nc \n');
            fprintf(fid,formatstr,CP1P2');
            fprintf(fid,'c \nc CP1\nc \n');
            fprintf(fid,formatstr,CP1');
            fprintf(fid,'c \nc CP2\nc \n');
            fprintf(fid,formatstr,CP2');
            fprintf(fid,'c \nc Co\nc \n');
            fprintf(fid,formatstr,Co');
            fclose(fid);
      else if (tbldim == 3)
            fid = fopen(asciiname,'w');
            fprintf(fid,'c \nc CP1P2P3\nc \n');
            fprintf(fid,formatstr,CP1P2P3');
            fprintf(fid,'c \nc CP1P2\nc \n');
            fprintf(fid,formatstr,CP1P2');
            fprintf(fid,'c \nc CP1P3\nc \n');
```

```
      fprintf(fid,formatstr,CP1P3');
      fprintf(fid,'c \nc CP2P3\nc \n');
      fprintf(fid,formatstr,CP2P3');
      fprintf(fid,'c \nc CP1\nc \n');
      fprintf(fid,formatstr,CP1');
      fprintf(fid,'c \nc CP2\nc \n');
      fprintf(fid,formatstr,CP2');
      fprintf(fid,'c \nc CP3\nc \n');
      fprintf(fid,formatstr,CP3');
      fprintf(fid,'c \nc Co\nc \n');
      fprintf(fid,formatstr,Co');
      fclose(fid);
else

end
end
```

APPENDIX II.

The CARL Software

1. Generic Routines

This section describes the generic subroutines required for the Control Allocation with Rate Limiting software and how to use them. It is assumed that the software is implemented in the "SimShell 1.5" aircraft simulation environment developed for this research. Otherwise, some of the global variables may be undefined, and some slight modifications may need to be made.

1.1 The Main CARL Executive

A. Usage

CONALLO is the main executive for the Control Allocation with Rate Limiting software. Its purpose is to check the failure status of all of the controls and readjust the allocatable control vector, determine if control allocation with rate limiting is to be used or just the global direct allocation method, and load appropriate control constraints and effectiveness data. It then calls the GET_FACET routine to allocate the controls, and if required, calls RESTORE_U to restore the controls to a desired configuration.

A.1 CONALLO

Function Prototype, SUBROUTINE
COMMON A_CVARS, SIMVARS, SIMPARS, FLAGS, ALLOCAT, ALLODIAGS

ConAllo takes no arguments. To invoke Control Allocation,

```
CALL CONALLO
```

CONALLO sets up the necessary parameters and interacts with the rest of the constrained control allocation software to allocate controls based on a commanded moment.

Common Definitions

A_CVARS	[global]	contains all of the aircraft global variables ("SimShell1.5" specific).
SIMVARS	[global]	contains all of the global simulation variables ("SimShell1.5" specific).
SIMPARS	[global]	contains all of the global simulation flags and parameters ("SimShell1.5" specific).
FLAGS	[global]	contains the simulation real-time flags ("SimShell1.5" specific).
ALLOCAT	[global]	Control Allocation with Rate Limiting Globals.
ALLODIAGS	[global]	Diagnostics globals for Control Allocation with Rate Limiting (Optional).

B. General Remarks

Although ConAllo declares all three of the simulation COMMON blocks, very few of the variables are required. The

variables that ConAllo accesses and/or changes are:

Variable	Type	Description
Trimming	LOGICAL	Status of Trimmer.
Do_Linrize	LOGICAL	Status of Linearization Utility.
Initialized	LOGICAL	Initialization flag.
Do_Diags	LOGICAL	Diagnostics flag.
DT	REAL	Sample period (sec).
MOMCMD(i)	REAL	Commanded moment vector (control generated).
OLDMOM(i)	REAL	Attained moment vector due to controls (previous iter.).
LMOM	REAL	Generic input to control allocation (Used in direct allocation mode only).
MMOM	REAL	Generic input to control allocation (Used in direct allocation mode only).
NMOM	REAL	Generic input to control allocation (Used in direct allocation mode only).
U(i)	REAL	Actual control deflections vector (deg).
PSAT	REAL	Control position saturation flag.
RSAT	REAL	% rate saturation for the hardest-driven control (Calculated by the restoring algorithms).
NCTRLS	INTEGER	Number of allocatable controls.
UCMD(i)	REAL	Commanded control deflection vector (deg).
IFAIL(i)	INTEGER	Control failure status vector.

The ALLOCAT Common block is broken down as follows:

Variable	Type	Description
M	INTEGER	Number of controls to allocate.
U1	INTEGER	Number of 1st facet defining control.
U2	INTEGER	Number of 2nd facet defining control.
RTYPE	INTEGER	Type of restoring logic 0 - none 1 - min norm, 2 - min drag.
IU(20)	INTEGER	Bookkeeping: UALLO(i) <-> U(IU(i)).
MOM(3)	REAL	Commanded moment inputs (deltas or global).
BMAT(3,20)	REAL	Control power matrix (local or global).
UMIN(20)	REAL	The minimum control deflection limits.
UMAX(20)	REAL	The maximum control deflection limits.
URMIN(20)	REAL	Deflection rate limits (toward the minimum pos. constraint).
URMAX(20)	REAL	Deflection rate limits (toward the maximum pos. constraint).
UALLO(20)	REAL	The allocated control deflections.
SATM	REAL	Saturation level (measured in moment space).
RSATU	REAL	Saturation level for the hardest driven control.

C. Functional Description

CONALLO begins by calculating the attained moments from the previous frame by formulating the global (slope at the origin) control effectiveness matrix through a call to A_C$GETUEFF, and multiplying the resulting matrix by the obtained control position vector. The result is stored in OLDMOM(). Next, the USE_GLOBALS flag is set based on whether the simulation is trimming, linearizing, or running. (In run mode, rate-limiting control allocation can be used and the flag is set to FALSE). CONALLO proceeds by checking the IFAIL flag for each allocatable control. If a control is reported as failed, then it is dropped from the list of controls to allocate. Bookkeeping is

maintained by the parameter vector IU() which maps the allocated control to its corresponding aircraft control. For example, if control 2 on the aircraft is reported as failed, then the second allocated control is actually the third aircraft control and so IU(2) = 3. The vector of failed controls and their mappings is also maintained in UFAIL() and IUF() so that the moment due to failed controls may be calculated. (Note that OLDMOM contains the general control generated moments for both failed and functional controls). After the allocatable control vector is resized and shifted as necessary, a check is performed to make sure that there are still 3 or more available controls. If not, then the ABORT flag (in the FLAGS COMMON block) is set to 1 and CONALLO stops execution.

Assuming there are sufficient controls to allocate, CONALLO constructs the effectiveness matrix BMAT using either global effectiveness if USE_GLOBALS is TRUE or local effectiveness if USE_GLOBALS is FALSE. This information is extracted from the A_C$GETUEFF function as before. Next, the constraint vectors are set using another aircraft-specific call to A_C$GETCSTR. The constraints returned are the minimum and maximum position constraints, and the rate limits for the minimum and maximum directions. If USE_GLOBALS is FALSE, CONALLO uses the position constraints and rate capabilities to determine the most restrictive limit as either the amount a control can move in one frame or the amount a control can move without violating its position constraints. Otherwise, CONALLO simply uses the returned position constraints. The input moments are then specified using either moment commands or the changes in commanded moment, depending once again on the state of the USE_GLOBALS flag. With knowledge of the constraints, CONALLO then sets the position saturation flag to 1 if any of the constraints are zero (indicating that a control is saturated). It also checks to see if the input moments are zero. If so, then no allocation is required and the ALLOCATED flag is set to TRUE

If the ALLOCATED flag is FALSE, then CONALLO continues by making a call to GET_FACET using the coordinates of the facet that worked in the previous frame (if a previous frame existed). If the controls could not be allocated, then it starts a facet search by calling GET_FACET with different combinations of the “2” controls until either a solution is found or no more facets are available to search. If there is enough rate capability remaining after direct allocation has been performed, control restoring can be invoked (but only if USE_GLOBALS is FALSE and the RTYPE parameter is not zero). The restored control vector is combined with the allocated control vector within the RESTORE_U subroutine. Finally, the commanded control vector is obtained by either adding the allocated controls to the current control positions (in the case of rate limiting allocation) or by simply taking the allocated control vector. If diagnostics is enabled, it also calls ALLODIAGSOUT to write the diagnostic information for the current frame.

D. Errors and Restrictions

When there are less than 3 controls, CONALLO sets the ABORT flag to 1, writes a message to the output window and halts execution. Any other errors that may occur within other control allocation modules are given an ID number and description. These can be found in the ISTATUS and MSG fields (or alternatively, the DG_ISTAT and DG_IMSG fields). In any event of a failure, CONALLO treats it as a “missed frame” and does not allocate controls. However, control restoring can still take place.

During real-time mode, when Control Allocation with Rate Limiting is active, the allocated change in controls vector is overdriven with a gain to compensate for the fact that CARL commands average deflection rates which may not be obtained by the actuators. This gain is based on a first order actuator model and will have to be changed if different models are to be used.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ---------------------------------------------------------------------------
```

```
!
!       Module Name: CONALLO
!           Called By: SHELL_CONALLO, TRIMMER, LINRIZE, SIMBATCH
!           Calls to: A_C$GETUEFF, GET_FACET, A_C$GETCSTR, RESTORE_U,
!                     ALLODIAGSOUT
!
! ------------------------------------------------------------------------
      SUBROUTINE CONALLO
! ------------------------------------------------------------------------
!
!       Function:   Main Control Allocation executive
!
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                           By
!  JUL 09 1996        Created. (Based on major revisions to version
!                     3.0)                                            JB
! ------------------------------------------------------------------------
!
!     Glossary Section
!
! ------------------------------------------------------------------------
!                             Global Variables
!
!     Name     |   Type    |           Description
*Trimming       LOGICAL     Status of Trimmer
*Do_Linrize     LOGICAL     Status of Linearization Utility
*Initialized    LOGICAL     Initialization flag
*Do_Diags       LOGICAL     Diagnostics flag
*DT             REAL        Time step (sec)
*MOMCMD(1)      REAL        Commanded Cl (control generated)
*OLDMOM(1)      REAL        Actual Cl due to controls (previous iter.)
*LMOM           REAL        Input to Control allocation (Cl or dCl)
*MMOM           REAL        Input to control allocation (Cm or dCm)
*NMOM           REAL        Input to control allocation (Cn or dCn)
*U(1)           REAL        Control 1 deflection (deg)
*PSAT           REAL        Control Position saturation flag
*RSAT           REAL        % control rate saturation
*NCTRLS         INTEGER     Number of configurable controls
*UCMD(1)        REAL        Control 1 def. Command (deg)
*IFAIL(1)       INTEGER     Failure Status: Control 1
*UTAU(1)        REAL        1st order time constant U(1)
!
!                             CONALLO Globals
!
!     Name     |   Type    |           Description
*MF             INTEGER     Number of failed controls
*M              INTEGER     Number of controls to allocate
*U1             INTEGER     Number of 1st control defining facet
```

```
*U2             INTEGER     Number of 2nd control defining facet
*IU(M)          INTEGER     Book keeping: UALLO(M) <-> U(IU(M))
*RTYPE          INTEGER     Type of restoring logic to use
*
*
*URMIN(M)       REAL        The deflection rate limits (min. direction)
*URMAX(M)       REAL        The deflection rate limits (max. direction)
*UMIN(M)        REAL        The minimum control deflection constraints
*UMAX(M)        REAL        The maximum control deflection constraints
*BMAT(3,M)      REAL        Control power matrix (local or global)
*UALLO(M)       REAL        The allocated control deflections
*MOM(3)         REAL        Commanded moment inputs (deltas or global)
*SATM           REAL        saturation level (taken in moment space)
*RSATU          REAL        maximum rate saturation level (control space)
*
!
!                             Local Variables
!
!     Name     |   Type    |           Description
*Allocated      LOGICAL     TRUE if allocation successful
*Isfailure      LOGICAL     TRUE if there is a failure present
*Use_Globals    LOGICAL     TRUE if global limits and eff. is to be used
*
*UFAIL(MF)      REAL        vector of failed controls
*IUF(MF)        INTEGER     Book keeping: UFAIL(MF) <-> U(IUF(MF))
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------
      REAL         A_CVARR  (250)
      CHARACTER*4  A_CVARC (100)
      INTEGER      A_CVARI  ( 20)
      CHARACTER*80 SIMPARC80( 10)
      LOGICAL      SIMPARL  ( 30)
      REAL         SIMVARR  (200)
      INTEGER      SIMVARI  ( 40)

      INTEGER Max_Controls
      PARAMETER (Max_Controls = 20)

      LOGICAL Trimming, Do_Linrize, Initialized, Do_Diags
      INTEGER IFAIL(Max_Controls)
      INTEGER NCTRLS
      REAL DT
      REAL MOMCMD(3), OLDMOM(3), LMOM, MMOM, NMOM, PSAT, RSAT
      REAL U(Max_Controls), UCMD(Max_Controls), UTAU(Max_Controls)
! --------------------------CONALLO Globals----------------------------
      REAL URMIN(Max_Controls), URMAX(Max_Controls),UMIN(Max_Controls),
```

```
     .     UMAX(Max_Controls) , BMAT(3,Max_Controls),
     .     UALLO(Max_Controls), MOM(3), SATM, RSATU
      INTEGER M, U1, U2, IU(Max_Controls), RTYPE
! -------------------------Diagnostic Records-------------------------
      INTEGER DG_ISTAT,DG_FACET(Max_Controls)
      LOGICAL DG_U_Globals
      REAL DG_ET,DG_MAXET
      CHARACTER*80 DG_IMSG
! --------------------------REALTIME Flags----------------------------
      INTEGER GEARDOWN,BRAKEON,AGILEVUON,ABORT
! -------------------------------Locals-------------------------------
      LOGICAL Allocated, Isfailure, Use_Globals
      INTEGER I, J, MF, IUF(Max_Controls), ISTATUS
      REAL UFAIL(Max_Controls),SECNDS, ETOTAU, GK
      CHARACTER*80 MSG
! --------------------------Shared Library----------------------------
      POINTER / REAL / ptr2A_CVARS
      REAL A_C$GETUEFF
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / A_CVARS / A_CVARR,A_CVARI,A_CVARC
      COMMON / SIMVARS / SIMVARR,SIMVARI
      COMMON / SIMPARS / SIMPARL,SIMPARC80

      COMMON / FLAGS   / GEARDOWN, BRAKEON, AGILEVUON, ABORT
      COMMON / ALLOCAT / M, U1, U2, RTYPE, IU, MOM, BMAT, UMIN, UMAX,
     .                   URMIN, URMAX, UALLO, SATM, RSATU
      COMMON / ALLODIAGS / DG_FACET,DG_ISTAT,DG_U_Globals,DG_ET,
     .                     DG_MAXET, DG_IMSG
! ---------------------------------------------------------------------
!
!     Equivalence Section
!
! ---------------------------------------------------------------------
      EQUIVALENCE (SIMPARL(15)   , Trimming       )
      EQUIVALENCE (SIMPARL(5)    , Do_Linrize     )
      EQUIVALENCE (SIMPARL(8)    , Initialized    )
      EQUIVALENCE (SIMVARR(1)    , DT             )
      EQUIVALENCE (A_CVARR(32)   , MOMCMD(1)      )
      EQUIVALENCE (A_CVARR(36)   , OLDMOM(1)      )
      EQUIVALENCE (A_CVARR(30)   , LMOM           )
      EQUIVALENCE (A_CVARR(31)   , MMOM           )
      EQUIVALENCE (A_CVARR(35)   , NMOM           )
      EQUIVALENCE (SIMVARR(111)  , U(1)           )
      EQUIVALENCE (A_CVARR(44)   , PSAT           )
      EQUIVALENCE (A_CVARR(53)   , RSAT           )
      EQUIVALENCE (A_CVARI(3)    , NCTRLS         )
      EQUIVALENCE (SIMVARR(131)  , UCMD(1)        )
```

```
      EQUIVALENCE (SIMVARI(11)   , IFAIL(1)       )
      EQUIVALENCE (SIMPARL(1)    , Do_Diags       )
      EQUIVALENCE (A_CVARR(144)  , UTAU(1)        )

! ----------------------------------------------------------------------
!
!     Initialization Section
!
! ----------------------------------------------------------------------

! ----------------------------------------------------------------------
!
!     Run Section
!
! ----------------------------------------------------------------------
      DG_ET = SECNDS(0.0)
      MF = 0
      Allocated = .FALSE.
      Isfailure = .FALSE.
      SATM = 0.
      RSATU = 0.
      ISTATUS = -99
      MSG = ' '

! Get location of the COMMON block A_CVARS (used for the
! ppc shared library interface)

      ptr2A_CVARS = %LOC(A_CVARR(1))

! Get the approximate control generated moments for current frame using
! the slope at the origin method.

      DO 1005 I = 1,3
        OLDMOM(I) = 0.0
        DO 1006 J = 1,NCTRLS
          BMAT(I,J) = A_C$GETUEFF(ptr2A_CVARS,I,J,0.0)
          OLDMOM(I) = OLDMOM(I) + BMAT(I,J)*U(J)
1006    CONTINUE
1005  CONTINUE

! Use local effectiveness and contraints during RUNTIME, and use the
! slopes at the control origins and global contraints during
! TRIM, LINEARIZATION.

      IF (Trimming .OR. Do_Linrize) THEN
        Use_Globals = .TRUE.
      ELSE
        Use_Globals = .FALSE.
      END IF
      DG_U_Globals = Use_Globals
```

```
! See if all of the controls are working and reconfigure the controls
! to allocate if we have to.

      J = 1
      M = NCTRLS
      DO 1020 I = 1,NCTRLS
        IF (IFAIL(I) .NE. 0) THEN
          Isfailure = .TRUE.
          M = M - 1
          MF = NCTRLS - M
          IUF(MF) = I
          UFAIL(MF) = U(I)
          IF (U1 .GT. M .OR. U2 .GT. M) THEN
!         the column for the control eff. of the control that worked last
!         frame has changed. We better search from scratch.
            U1 = 0
            U2 = 0
          END IF
        ELSE
          IU(J) = I
          J = J + 1
        END IF
1020  CONTINUE

      DO 1025 I = 1,M
        UALLO(I) = 0.0
1025  CONTINUE

! Can we still allocate? If not, get out.

      IF (M .LT. 3) THEN
        WRITE(6,'(1x,A)') 'TOO FEW CONTROLS TO ALLOCATE--EJECT! EJECT!'
        ABORT = 1
        RETURN
      END IF

! Get the control power matrices for the controls (that work).

      DO 1030 I = 1,3
        DO 1031 J = 1,M
          IF (Use_Globals) THEN
            BMAT(I,J) = A_C$GETUEFF(ptr2A_CVARS,I,IU(J),0.0)
          ELSE
            BMAT(I,J) = A_C$GETUEFF(ptr2A_CVARS,I,IU(J),U(IU(J)))
          END IF
1031    CONTINUE
1030  CONTINUE

! Set the control minimum and maximum contraints for allocation. This
! is done by getting the position limits and rate limits and then taking
! the most restrictive of either the position limit or the amount that
```

```
! a control can move in one frame (for run mode only, otherwise, we
! take the position limits)

      CALL A_C$GETCSTR(ptr2A_CVARS, M, IU, UMAX, UMIN, URMAX, URMIN)

      IF (.NOT. Use_globals) THEN
        DO 1035 I = 1,M
          UMIN(I) = AMIN1(AMAX1((UMIN(I) - U(IU(I))),-URMIN(I)*DT),0.0)
          UMAX(I) = AMAX1(AMIN1((UMAX(I) - U(IU(I))), URMAX(I)*DT),0.0)
1035    CONTINUE
      END IF

! Input moment commands. (we use absolute moment commands when
! TRIMMING. For RUNMODE, we use "delta" moment commands.)

      IF (Use_Globals) THEN
        MOM(1) = LMOM
        MOM(2) = MMOM
        MOM(3) = NMOM
      ELSE
        DO 1042 I = 1,3
          MOM(I) = MOMCMD(I) - OLDMOM(I)
1042    CONTINUE
      END IF

! Check for any position saturation of controls

      PSAT = 0.0
      DO 1050 I = 1,M
        IF (UMAX(I) .EQ. 0.0 .OR. UMIN(I) .EQ. 0.0) THEN
          PSAT = 1.0
        END IF
1050  CONTINUE

! Check to see if we even need to allocate or not

      IF (MOM(1) .EQ. 0. .AND. MOM(2) .EQ. 0. .AND. MOM(3) .EQ. 0.) THEN
        Allocated = .TRUE.
      END IF

      IF (.NOT. Allocated) THEN

! Start Control Allocation. We check the facet that worked last frame
! right now.

        IF (U1 .NE. 0 .AND. U2 .NE. 0) THEN
          CALL GET_FACET(UALLO, Allocated, SATM, ISTATUS, MSG,
     .                   BMAT,U1, U2, UMIN, UMAX, MOM, M)

          IF (Allocated) THEN
            GO TO 1059
```

```
          END IF
        END IF

! Oh man! now we have to start searching from scratch.

        DO 1051 U1 = 1,M-1
          DO 1052 U2 = U1+1,M
            CALL GET_FACET(UALLO, Allocated, SATM, ISTATUS,
     .                     MSG,BMAT,U1, U2, UMIN, UMAX, MOM, M)

            IF (Allocated) THEN
              GO TO 1059
            END IF

1052      CONTINUE
1051    CONTINUE

      END IF

1059  CONTINUE                    ! we're done allocating

      IF (.NOT. Use_Globals) THEN

        RSAT = SATM               ! rate saturation (moment space)

! Time for some control restoring algorithms

        IF (RTYPE .GT. 0 .AND. RSAT .LT. 1.0) THEN
          CALL RESTORE_U (MOMCMD, U)
        END IF

! Get allocated Control commands. (The GK factor is the gain required
! to overdrive the control commands so that actuator position = com-
! manded position

        DO 1060 I = 1,M
          ETOTAU = EXP(DT/UTAU(IU(I)))
          GK = ETOTAU/(ETOTAU - 1.0)
          UCMD(IU(I)) = U(IU(I)) + GK*UALLO(I)
1060    CONTINUE

      ELSE

! Here we are dealing with global positions not deltas

        PSAT = SATM          ! Position saturation (moment space)

        DO 1070 I = 1,M
          UCMD(IU(I)) = UALLO(I)
1070    CONTINUE
```

```
      END IF

      DG_ET = SECNDS(DG_ET)
      DG_ET = AMAX1(0.0,DG_ET)
      DG_ISTAT = ISTATUS
      DG_IMSG = MSG
      IF (Do_Diags) THEN
        CALL ALLODIAGSOUT
      END IF
! ---------------------------------------------------------------------
!
!     End of CONALLO
!
! ---------------------------------------------------------------------
      RETURN
      END
```

1.2 CARL Diagnostic Output

A. Usage

When diagnostics is enabled, CONALLO can write some of its internal results to a data file, thus allowing easier debugging and testing.

A.1 ALLODIAGSOUT

Function Prototype, SUBROUTINE

COMMON A_CVARS, SIMVARS, SIMPARS, ALLOCAT, ALLODIAGS

ALLODIAGSOUT takes no arguments

CALL ALLODIAGSOUT

Internal CONALLO information is written to two text files, CONALLO_DIAGS.TXT, and CONALLO_TIMING.TXT

Common Definitions

A_CVARS	[global]	contains all of the aircraft global variables (“SimShell1.5” specific).
SIMVARS	[global]	contains all of the global simulation variables (“SimShell1.5” specific).
SIMPARS	[global]	contains all of the global simulation flags and parameters (“SimShell1.5” specific).
ALLOCAT	[global]	Control Allocation with Rate Limiting Globals.
ALLODIAGS	[global]	Diagnostics globals for Control Allocation with Rate Limiting (Optional).

B. General Remarks

A sample of the type of diagnostic output that is generated when ALLODIAGSOUT is called is shown below:

File CONALLO_DIAGS.TXT

```
Simulation Time:  0.1000 sec Use_Globals: F
Failure Status:  0      0     0     0     0     0     0     0     0
Control Position: 1.31   -1.48    2.23   -2.05   1.15  -1.19  1.51  -2.03  -1.64

Moment Commands: -0.0049      0.0032     -0.0006

Control Eff. Matrix:
 0.0008 -0.0008  0.0006 -0.0006  0.0001 -0.0001   0.0001  0.0000  0.0000
-0.0057 -0.0056 -0.0001  0.0001  0.0001  0.0030   0.0028  0.0000 -0.0021
 0.0004 -0.0005  0.0001 -0.0001 -0.0014  0.0004  -0.0004 -0.0026  0.0000

Control Constraints
MIN:  -2.25      -2.25    -4.50   -4.50    -6.75    -3.75    -3.75    -4.00  -4.00
MAX:   2.25       2.25     4.50    4.50     6.75     3.75     3.75     4.00   4.00

Allocation:
  FACET CODE:  0  1     0     1     2     1     0     1     2
  ISTAT:  0      NORMAL
```

```
   RSAT:    0.51
   UALLO:  -0.97   0.90   -2.16    2.13   -2.92   1.93  -1.91  2.04   -0.93

 Restore Type:  1
 Commanded Position: 0.34  -0.59   0.07  0.08  -1.78   0.74  -0.40   0.01  -2.57

 Execution Time: 0.0162
```

For each frame, ALLODIAGSOUT writes the failure status of the controls, the current control positions and control generated moments, the control effectiveness matrix for the functional controls, (USE_GLOBALS determines if the data is global or local), the current control constraints for the functional controls, (Once again this data depends on USE_GLOBALS), and the allocation results, which include the Facet code for which controls were allocated to, the error status and description, the amount of rate saturation, and the allocated control vector. The commanded control vector is displayed (which depends on the restoring type), and the time required to allocate is reported. The CONALLO_TIMING.TXT file contains only the time required to allocate for each frame.

C. Functional Description

The steps that ALLODIAGSOUT takes depend on the two parameters BATRUN and DO_DIAGS, (located in the SIMPARS common block). When diagnostics is enabled and before the simulation begins running, the batch mode simulation loop makes an initialization call to ALLODIAGSOUT. Since BATRUN is FALSE and DO_DIAGS is TRUE, ALLODIAGSOUT initializes the two diagnostic files. Once the simulation loop starts, BATRUN is TRUE, and the path taken by ALLODIAGSOUT is to write the diagnostic output. Once the simulation stops execution, the diagnostic output files must be closed by making a call to ALLODIAGSOUT with DO_DIAGS set to false. In this case, the logical file units are closed and ALLODIAGSOUT returns.

D. Errors and Restrictions

When using the diagnostic output routines in the Shell, it is important to close the files before they are viewed. (This is done by calling any diagnostic routine with DO_DIAGS temporarily set to FALSE). Ideally, any module utilizing the diagnostics feature should contain a diagnostic “DUMP” or similar command to do this automatically. Be warned that diagnostic output consumes a great deal of processor time due to all of the data I/O. Its use should be avoided for real-time simulations.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: ALLODIAGSOUT
!           Called By: CONALLO
!           Calls to: none
!
! ------------------------------------------------------------------------
      SUBROUTINE ALLODIAGSOUT
! ------------------------------------------------------------------------
!
!       Function:   Writes the ConAllo diagnostic output to the file
!                CONALLO_DIAGS.TXT.
!
```

```
!
! ------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                          By
!  JAN 11 1997        Created                                      J.B.
!
! ------------------------------------------------------------------
!
!     Glossary Section
!
! ------------------------------------------------------------------
!                              Global Variables
!
!     Name     |   Type    |           Description
*Do_Diags       LOGICAL     Diagnostics flag
*BatRun         LOGICAL     Simulation running in batch mode
*T              REAL        Time (sec)
*IFAIL(1)       INTEGER     Failure Status: Control 1
*U(1)           REAL        Control 1 deflection (deg)
*NCTRLS         INTEGER     Number of configurable controls
*RSAT           REAL        % control rate saturation
*UCMD(1)        REAL        Control 1 def. Command (deg)
!
!                              Local Variables
!
!     Name     |   Type    |           Description
*VARNAME        VARTYPE     VARDESCRIPTION
! ------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------
      REAL         A_CVARR  (250)
      CHARACTER*4  A_CVARC (100)
      INTEGER      A_CVARI  ( 20)
      CHARACTER*80 SIMPARC80( 10)
      LOGICAL      SIMPARL  ( 30)
      REAL         SIMVARR  (200)
      INTEGER      SIMVARI  ( 40)

      INTEGER Max_Controls
      PARAMETER (Max_Controls = 20)
      LOGICAL Do_Diags, BatRun
      REAL T, U(20), UCMD(20)
      INTEGER IFAIL(20), NCTRLS, I, J
      CHARACTER*1 TAB

! -------------------------Diagnostic Records-------------------------
```

```
      INTEGER DG_ISTAT,DG_FACET(Max_Controls)
      LOGICAL DG_U_Globals
      REAL DG_ET,DG_MAXET
      CHARACTER*80 DG_IMSG

! ---------------------------CONALLO Globals----------------------------
      REAL URMIN(Max_Controls), URMAX(Max_Controls),UMIN(Max_Controls),
     .     UMAX(Max_Controls) , BMAT(3,Max_Controls),
     .     UALLO(Max_Controls), MOM(3), SATM, RSATU
      INTEGER M, U1, U2, IU(Max_Controls), RTYPE
! ----------------------------------------------------------------------
!
!     Common Section
!
! ----------------------------------------------------------------------
      COMMON / A_CVARS / A_CVARR,A_CVARI,A_CVARC
      COMMON / SIMVARS / SIMVARR,SIMVARI
      COMMON / SIMPARS / SIMPARL,SIMPARC80

      COMMON / ALLOCAT / M, U1, U2, RTYPE, IU, MOM, BMAT, UMIN, UMAX,
     .                   URMIN, URMAX, UALLO, SATM, RSATU
      COMMON / ALLODIAGS / DG_FACET,DG_ISTAT,DG_U_Globals,DG_ET,
     .                     DG_MAXET, DG_IMSG
! ----------------------------------------------------------------------
!
!     Equivalence Section
!
! ----------------------------------------------------------------------
      EQUIVALENCE (SIMPARL(1)    , Do_Diags       )
      EQUIVALENCE (SIMPARL(17)   , BatRun         )
      EQUIVALENCE (SIMVARR(2)    , T              )
      EQUIVALENCE (SIMVARI(11)   , IFAIL(1)       )
      EQUIVALENCE (SIMVARR(111)  , U(1)           )
      EQUIVALENCE (A_CVARI(3)    , NCTRLS         )
      EQUIVALENCE (SIMVARR(131)  , UCMD(1)        )

! ----------------------------------------------------------------------
!
!     Initialization Section
!
! ----------------------------------------------------------------------
      TAB = CHAR(9)
      IF (.Not. BatRun) THEN
        IF (Do_Diags) THEN
          OPEN (UNIT = 66, FILE = 'CONALLO_DIAGS.TXT', STATUS = 'NEW')
          WRITE(66,101)
          OPEN (UNIT = 67, FILE = 'CONALLO_TIMING.TXT', STATUS = 'NEW')
          WRITE(67,201)
          WRITE (67,202) TAB
        ELSE
          CLOSE (66)
```

```
          CLOSE (67)
          RETURN
        END IF
      END IF
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      WRITE (66,102) T,DG_U_Globals
      WRITE (66,103) (IFAIL(I),TAB, I = 1,NCTRLS)
      WRITE (66,104) (U(I),TAB, I = 1,NCTRLS)

      WRITE (66,105) (MOM(I),TAB, I = 1,3)

      WRITE (66,106)
      DO 1010 I = 1,3
        WRITE (66,107) (BMAT(I,J),TAB, J = 1,M)
1010  CONTINUE

      WRITE (66,108)
      WRITE (66,109) (UMIN(I),TAB, I = 1,M)
      WRITE (66,110) (UMAX(I),TAB, I = 1,M)

      WRITE (66,111)
      WRITE (66,112) (DG_FACET(I),TAB, I = 1,M)
      WRITE (66,113) DG_ISTAT, TAB, DG_IMSG
      WRITE (66,114) SATM
      WRITE (66,115) (UALLO(I),TAB, I = 1,M)

      WRITE (66,116) RTYPE
      WRITE (66,117) (UCMD(I),TAB, I = 1,NCTRLS)

      WRITE (66,118) DG_ET

      WRITE (67,203) T,TAB,DG_ET

101   FORMAT (1x,'ConAllo Diagnostic Output'/)
102   FORMAT (/,1x,'Simulation Time: ',F7.4,' sec ','Use_Globals: ',L1)
103   FORMAT (1x,'Failure Status: ',<NCTRLS>(I2,A1))
104   FORMAT (1x,'Control Position:   ',<NCTRLS>(F10.6,A1),/)
105   FORMAT (1x,'Moment Commands: ',3(E15.8,A1),/)
106   FORMAT (1X,'Control Eff. Matrix:')
107   FORMAT (1X,<M>(E15.8,A1))
108   FORMAT (/,1X,'Control Constraints')
109   FORMAT (1X,'MIN: ',<M>(F8.4,A1))
110   FORMAT (1X,'MAX: ',<M>(F8.4,A1),/)
111   FORMAT (1X,'Allocation:')
112   FORMAT (3X,'FACET CODE: ',<M>(I2,A1))
113   FORMAT (3X,'ISTAT: ',I3,A1,A)
114   FORMAT (3X,'Saturation:  ',F6.2)
```

```
115   FORMAT (3X,'UALLO: ',<M>(F8.4,A1),/)
116   FORMAT (1X,'Restore Type: ',I2)
117   FORMAT (1X,'Commanded Position: ',<NCTRLS>(F8.4,A1),/)
118   FORMAT (1X,'Execution Time: ',F6.4,/)

201   FORMAT ('*')
202   FORMAT ('Sim Time',A1,'Elapsed Time')
203   FORMAT (F7.4,A1,F6.4)
! ---------------------------------------------------------------------
!
!     End of ALLODIAGSOUT
!
! ---------------------------------------------------------------------
      RETURN
      END
```

1.3 Control Restoring Techniques

A. Usage

These algorithms can be used by control allocation with rate limiting to restore the controls to some desired position while attaining desired moments.

A.1 RESTORE_U

Function Prototype, SUBROUTINE

REAL	MOMCMD(3), U(M),RSAT
COMMON	A_CVARS, ALLOCAT

Assign values to MOMCMD and U

CALL RESTORE_U(MOMCMD, U, RSAT)

RESTORE_U combines the restored control deflections with the allocated deflections in UALLO, and returns the percent of rate saturation in RSAT

Argument Definitions

MOMCMD(3)	[in]	The Commanded moment vector for the current frame.
U(M)	[in]	The vector of current aircraft control positions
RSAT	[out]	The highest amount of rate saturation calculated for all of the controls
ALLOCAT	[global]	ConAllo globals

B. General Remarks

The current implementation of RESTORE_U provides 2 restoring methods. When the RTYPE flag in the ALLOCAT common block is 1, minimum norm restoring is used. The objective function that is minimized in this case is F = [(π/180) ð]/2. When RTYPE is 2, minimum drag restoring is used. The gradient of F for this case is taken as the drag effectiveness of each of the controls.

C. Functional Description

Both restoring methods use the same restoring algorithm. The only difference is in the data that is used to augment the 4th row of the control effectiveness matrix. When RTYPE is 1, RESTORE_U augments the control effectiveness matrix BMAT with a 4th row corresponding to $\partial F/\partial u$, (2*(π/180) ð). Otherwise, if RTYPE is 2 then it augments BMAT with the controls' drag effectiveness. RESTORE_U then proceeds to create the objective vector DELO = $(0,0,0,-1)^T$ and calculates the pseudo-inverse of the augmented 4xm BMAT with a call to PINVB4. Using this matrix, a solution is found for $\Delta u = B^T[BB^T]^{-1}DELO$. Since the pseudo-inverse solution has no knowledge of the control constraints, it is checked to make sure that none are violated. In the case of a constraint violation, a scaling factor is found such that when the solution vector is uniformly scaled, the offending control is just at the point of saturation. An additional minimization factor of 0.1 is applied to the scaled solution vector. Finally, the Δu vector found above is combined with the allocated control vector, and the % rate saturation of the hardest driven control is calculated and returned in RSAT.

D. Errors and Restrictions

In some cases (like minimum drag restoring at high angles of attack), the effectiveness data indicates that the minimum objective occurs at a control constraint. As a consequence, RESTORE_U will attempt to drive the controls toward these positions. This may have adverse effects on the available maneuverability of the aircraft and should therefore be avoided.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: RESTORE_U
!           Called by: CONALLO
!           Calls to: PINVB4, A_C$GETUEFF
!
! ------------------------------------------------------------------------
      SUBROUTINE RESTORE_U (MOMCMD, U)
! ------------------------------------------------------------------------
!
!       Function:    Performs various control-restoring techniques
!                 according to RTYPE.
!
!       RTYPE = 0 -> No Restoring
!       RTYPE = 1 -> Restore towards the Minimum Norm solution.
!       RTYPE = 2 -> Restore towards the min CD solution.
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                          By
!  JUL 09 1996        Created,                                        JB
!  OCT 21 1996        Rewrote the minimum-norm restoring logic. Instead
!                     of using the null-space projection method of the
!                     pseudo inverse, we use a least squares approach
!                     by specifying a function of the the squares of the
!                     controls.
!  MAR 23 1997        Removed the RSAT argument (Now defined in ALLOCAT
!                     Common block)                                   JB
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------

      INTEGER Max_Controls
      PARAMETER (Max_Controls = 20)
! -------------------------CONALLO Globals----------------------------
      REAL URMIN(Max_Controls), URMAX(Max_Controls),UMIN(Max_Controls),
     .     UMAX(Max_Controls) , BMAT(3,Max_Controls),
     .     UALLO(Max_Controls), MOM(3), SATM, RSATU
```

```
      INTEGER M, U1, U2, IU(Max_Controls), RTYPE
! ------------------------------Locals---------------------------------
      REAL UP(Max_Controls), DU(Max_Controls), U(Max_Controls),
     .     PMAT(Max_Controls,3), MOMCMD(3), RSATO, SC, SC1
      INTEGER I,J
      REAL P4(Max_Controls,4), ROW4(Max_Controls),
     .     B4MAT(4,Max_Controls), DELO(4)
! --------------------------Shared Library-----------------------------
      POINTER / REAL / ptr2A_CVARS
      REAL A_C$GETUEFF
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / ALLOCAT / M, U1, U2, RTYPE, IU, MOM, BMAT, UMIN, UMAX,
     .                   URMIN, URMAX, UALLO, SATM, RSATU

! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      IF (RTYPE .EQ. 1) THEN
! ---------------------------------------------------------------------
! Minimum-Norm restoring
! ---------------------------------------------------------------------

! Get the 4th row of the B matrix (Y' = .001745*U)

        DO 1012 I=1,3
          DO 1013 J=1,M
            B4MAT(I,J) = BMAT(I,J)
1013      CONTINUE
1012    CONTINUE
        DO 1014 J=1,M
          B4MAT(4,J) = 1.74533E-3*U(IU(J))
1014    CONTINUE

! make the Δobjective vector (0,0,0,-1)^t

      DELO(1) = 0.
      DELO(2) = 0.
      DELO(3) = 0.
      DELO(4) = -1.0

      CALL PINVB4(M, B4MAT, P4)

! find a solution Δu satisfying Δu = P4*DELO

      DO 1015 I = 1,M
```

```
        DU(I) = 0.
        DO 1016 J = 1,4
          DU(I) = DU(I) + P4(I,J)*DELO(J)
1016    CONTINUE
1015  CONTINUE

! now check that the pseudo inverse solution has not violated constraints
! and fix if necesary

        SC = 1.
        DO 1020 I=1,M
          SC1 = SC
          IF (DU(I) .GT. (UMAX(I) - UALLO(I))) THEN
              SC1 = (UMAX(I) - UALLO(I))/DU(I)
          END IF
          IF (DU(I) .LT. (UMIN(I) - UALLO(I))) THEN
              SC1 = (UMIN(I) - UALLO(I))/DU(I)
          END IF
          SC = AMIN1(SC,SC1)
1020    CONTINUE

! apply minimization factor to the scale factor also and scale controls

      DO 1025 I = 1,M
        DU(I) = 0.1*SC*DU(I)
1025  CONTINUE

! Calculate the restored controls and rate saturation

        RSATU = 0.
        DO 1030 I=1,M
          RSATO = RSATU
          UALLO(I) = UALLO(I) + DU(I)
          IF (UALLO(I) .LT. 0.) THEN
            RSATU = 1. - (UMIN(I)-UALLO(I))/UMIN(I)
          ELSE
            IF (UALLO(I) .GT. 0.) THEN
              RSATU = 1. - (UMAX(I)-UALLO(I))/UMAX(I)
            ELSE
              RSATU = 0.
            END IF
          END IF
          RSATU = AMAX1(RSATU,RSATO)
1030    CONTINUE

      END IF
! ---------------------------------------------------------------------
! End of Minimum-norm Restoring
! ---------------------------------------------------------------------
      IF (RTYPE .EQ. 2) THEN
! ---------------------------------------------------------------------
```

```
! Minimum drag restoring
! ---------------------------------------------------------------------
        ptr2A_CVARS = %LOC(A_CVARR(1))

! Get the 4th row of the B matrix (corresponding to drag)

        DO 1952 I=1,3
          DO 1951 J=1,M
            B4MAT(I,J) = BMAT(I,J)
1951      CONTINUE
1952    CONTINUE
        DO 1953 J=1,M
          B4MAT(4,J) = A_C$GETUEFF(ptr2A_CVARS,4,IU(J),U(IU(J)))
1953    CONTINUE

! make the Δobjective vector (0,0,0,-1)^t

      DELO(1) = 0.
      DELO(2) = 0.
      DELO(3) = 0.
      DELO(4) = -1.0

      CALL PINVB4(M, B4MAT, P4)

! find a solution Δu satisfying Δu = P4*DELO

      DO 1060 I = 1,M
        DU(I) = 0.
        DO 1061 J = 1,4
          DU(I) = DU(I) + P4(I,J)*DELO(J)
1061    CONTINUE
1060  CONTINUE


! now check that the pseudo inverse solution has not violated constraints
! and fix if necesary

      SC = 1.
      DO 1066 I=1,M
        SC1 = SC
        IF (DU(I) .GT. (UMAX(I) - UALLO(I))) THEN
            SC1 = (UMAX(I) - UALLO(I))/DU(I)
        END IF
        IF (DU(I) .LT. (UMIN(I) - UALLO(I))) THEN
            SC1 = (UMIN(I) - UALLO(I))/DU(I)
        END IF
        SC = AMIN1(SC,SC1)
1066  CONTINUE

! apply minimization factor to the scale factor also and scale controls
```

```
      DO 1067 I = 1,M
        DU(I) = 0.1*SC*DU(I)
1067  CONTINUE

! Calculate the restored controls and rate saturation

      RSATU = 0.
      DO 1068 I=1,M
        RSATO = RSATU
        UALLO(I) = UALLO(I) + DU(I)
        IF (UALLO(I) .LT. 0.) THEN
          RSATU = 1. - (UMIN(I)-UALLO(I))/UMIN(I)
        ELSE
          IF (UALLO(I) .GT. 0.) THEN
            RSATU = 1. - (UMAX(I)-UALLO(I))/UMAX(I)
          ELSE
            RSATU = 0.
          END IF
        END IF
        RSATU = AMAX1(RSATU,RSATO)
1068  CONTINUE

      END IF
! ---------------------------------------------------------------------
! End of minimum drag Restoring
! ---------------------------------------------------------------------
! ---------------------------------------------------------------------
!
!      End of RESTORE_U
!
! ---------------------------------------------------------------------
       RETURN
       END
```

1.4 Defining Facet Geometries

A. Usage

These subroutines calculate the geometry of the Attainable Moment Subset (AMS) facets given a pair of "face-defining" controls, their maximum and minimum constraints, and the control effectiveness matrix.

A.1 GET_FACET

Function Prototype, SUBROUTINE

REAL	UALLO(M), RSAT, BMAT(3,M), UMIN(M), UMAX(M), MOM(3)
INTEGER	ISTATUS, U1, U2, M
LOGICAL	ALLOCATED
CHARACTER*80	MSG

Assign data to BMAT, UMIN, UMAX, MOM, U1, U2, and M

CALL GET_FACET(UALLO, ALLOCATED, RSAT, ISTATUS, MSG, BMAT, U1, U2, UMIN, UMAX, MOM, M)

GET_FACET sets up the facet geometry pertaining to controls U1 and U2 and determines the required positions of the remaining controls. It then calls GET_MAT and GET_U in an attempt to allocate controls for the U1/U2 defined facets.

Argument Definitions

UALLO(M)	[out]	Allocated control vector (global or delta)
ALLOCATED	[out]	True if controls were allocated, otherwise false
RSAT	[out]	amount of rate saturation (of the ΔAMS or AMS)
ISTATUS	[out]	Status code of the allocation procedure
MSG	[out]	description of the ISTATUS code
BMAT(3,M)	[in]	The Control effectiveness matrix (global or local)
U1	[in]	number of the first facet defining control
U2	[in]	number of the second facet defining control
UMIN(M)	[in]	vector of minimum control constraints
UMAX(M)	[in]	vector of maximum control constraints
MOM(3)	[in]	vector of desired moments (global or delta)
M	[in]	number of controls to allocate
ALLODIAGS	[global]	Diagnostics globals for ConAllo

A.2 GET_MAT

Function Prototype, SUBROUTINE

REAL	UR(M), BMAT(3,M), UMIN(M), UMAX(M), MAT(3,3)
INTEGER	U1, U2, M

Assign data to BMAT, UMIN, UMAX, UR, U1, U2, and M

CALL GET_MAT(M, MAT, BMAT, UR, U1, U2, UMIN, UMAX)

GET_MAT calculates the current facet geometry and saves it in MAT

Argument Definitions

M	[in]	The number of controls to allocate
MAT	[out]	A matrix containing the current facet geometry
BMAT(3,M)	[in]	The Control effectiveness matrix (global or local)
UR(M)	[in]	The vector of required control positions that generate the current facet.
U1	[in]	number of the first facet defining control
U2	[in]	number of the second facet defining control
UMIN(M)	[in]	vector of minimum control constraints
UMAX(M)	[in]	vector of maximum control constraints

B. General Remarks

The code in GET_FACET and GET_MAT was originally taken from some of the early AMS drawing utilities. However, over a few years of development, it has lost most of its graphical functionality and now serves primarily as a preprocessor for the heart and soul of the control allocation software, GET_U. The current implementation has moved the calls to GET_U inside GET_FACET. This is done so that facet geometries can be calculated and checked one at a time, allowing the search to stop once the controls have been allocated. In addition, the geometry of the entire AMS is not built up as it was in the past.

C. Functional Description

The most important function of GET_FACET is the process of finding the outer-most facets of the AMS given a pair of face-defining controls, and determining the required positions of the other controls on these facets. The theory behind this implementation is based on the discussion in reference 3. However, the current algorithm used to find the transformation matrix has been modified. GET_FACET begins by finding the normal to the two columns associated with U1 and U2, and proceeds to find the magnitude of the resulting normal vector. If the magnitude of this vector is zero, (indicating that the two controls do not describe a face in moment space), then GET_FACET returns with an error code of 4. Otherwise, it finds the first row of a transformation matrix using the direction cosines of the normal vector (ie. the normal divided by its magnitude), such that when multiplied by the control effectiveness matrix, produces zeros in the first row entries corresponding to controls U1 and U2. In other words, A rotation is found such that the faces defined by controls U1 and U2 are perpendicular to the 1st axis in the rotated moment space. Next, the first row of the rotated B matrix is calculated by multiplying BMAT by the known 1st row of the transformation matrix. By inspecting the signs of these entries, the facet coordinates for the “positive” and “negative” (opposite) facets can be defined. Positive entries receive 1’s (implying maximum deflection), negative entries receive 0’s (minimum deflections), and the entries associated with U1 and U2 (the varying controls), are given 2’s. At the same time, the opposite facet is defined as having the opposite coordinates given by the positive facet. In some instances, other control pairs may define perpendicular faces as well, resulting in more than two zeros in the first row of the transformed B matrix (these are termed “special” controls). The required facet coordinates of the special controls are not easily determined, and so each possible combination of minimum and maximum deflection must be checked. Next, the positive facet geometry is defined in moment space through a call to GET_MAT, followed by a call to GET_U to check the facet and allocate controls. If the facet fails the tests in GET_U, then the opposite facet geometry is defined in moment space and sent to GET_U for checking and allocation. Control is then returned back to ConAllo.

GET_MAT takes the minimum or maximum control positions in the vector UR, changes the controls associated with U1 and U2 to their minimum values (0,0), and pre-multiplies by BMAT, giving the vertex of the U1/U2 defining facet in moment space with respect to the origin (M0). Next, it changes the U2 control to its maximum value (0,1) and multiplies by BMAT, giving another vertex of the U1/U2 defining facet (M1). Finally, the U2 control is reset to its minimum, the U1 control is set to its maximum (1,0), and the UR vector is multiplied by BMAT once again. This produces yet a 3rd vertex in moment space w.r.t. the origin (M2). The facet geometry in

moment space is then stored in a matrix as follows: Column 1 contains the 1st vertex with controls U1 and U2 set to their minimum values (M0), referenced from the origin, column 2 contains the vector from M0 to M2, and column 3 contains the vector from M0 to M1.

D. Errors and Restrictions

For the case when extra zeros appear in the rotated control effectiveness matrix in GET_FACET (indicating that other pairs of controls generate perpendicular faces besides the U1 and U2 controls), each combination of minimum and maximum deflection for each of these redundant controls must be checked. This is done within a loop in which GET_MAT and GET_U is repeatedly called. In the current implementation, the maximum number of redundant controls allowed is 4. If the maximum number of redundant controls is exceeded, GET_FACET returns with an error code of 5.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: GET_FACET
!           Called By: CONALLO
!           Calls to: GET_MAT, GET_U
!
! ------------------------------------------------------------------------
      SUBROUTINE GET_FACET(UALLO, Allocated, RSAT, ISTATUS, MSG,
     .                     BMAT, U1, U2, UMIN, UMAX, MOM, M)
! ------------------------------------------------------------------------
!
!       Function:   This subroutine generates and finds a valid FACET
!                 according to a given pair of controls and effectiveness
!                 matrix and sends its geometry to GET_U for control
!                 allocation.
!
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                          By
!  APR 15 1997        Added logic to test all the possible 0 and 1
!                     combinations for any "special" controls (any
!                     controls which form faces perp. to the U1 and
!                     U2 controls) The logic that does this is pretty
!                     slick and can handle any number of special con-
!                     trols. However the number of combinations gets
!                     really big as the # of special controls gets
!                     large. So I included the MAXS parameter to set
!                     the maximum number of special cases that will be
!                     checked before we just declare a missed frame.
!                     Right now MAXS is set to 4 special controls.    JB
!  JUL 09 1996        Created                                  J. Bolling
!
```

```
! ------------------------------------------------------------------
!
!     Glossary Section
!
! ------------------------------------------------------------------
!                             Global Variables
!
!     Name     |   Type    |           Description
*VARNAME        VARTYPE     VARDESCRIPTION
!
!                             Local Variables
!
!     Name     |   Type    |           Description
*U1             INTEGER     Number of 1st control defining facet
*U2             INTEGER     Number of 2nd control defining facet
*M              INTEGER     Number of controls to allocate
*MOM(3)         REAL        Commanded moment inputs (deltas or global)
*BMAT(3,M)      REAL        Control power matrix
*UALLO(M)       REAL        Vector of Allocated Controls
*Allocated      LOGICAL     Successful allocation flag
*RSAT           REAL        Rate Saturation level
*ISTATUS        INTEGER     Error Status Identifier
*MSG            CHARACTER   motivational message
*UMIN(M)        REAL        Minimum control constraints
*UMAX(M)        REAL        Maximum control constraints
! ------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------
      INTEGER U1, U2, M
      INTEGER Max_Controls
      PARAMETER (Max_Controls = 20)
      REAL MOM(3), BMAT(3,Max_Controls), UALLO(Max_Controls),
     .     UMIN(Max_Controls), UMAX(Max_Controls), RSAT
      INTEGER ISTATUS
      LOGICAL Allocated
      CHARACTER*80 MSG
! -------------------------Diagnostic Records-----------------------
      INTEGER DG_ISTAT,DG_FACET(Max_Controls)
      LOGICAL DG_U_Globals
      REAL DG_ET,DG_MAXET
      CHARACTER*80 DG_IMSG
! -------------------------------Locals-----------------------------
      INTEGER IMAXFAC(Max_Controls), IMINFAC(Max_Controls),
     .        IS(Max_Controls), I, J, NS, K, IN(Max_Controls), NN, DIV,
     .        BIT
      INTEGER MAXS
      PARAMETER (MAXS = 4)
```

```
      REAL BROW(Max_Controls), MAT_MAX(3,3), MAT_MIN(3,3), NORMAL(3),
     .     NORMMAG, A(2,2), AINV(2,2), TR1(3), UMAXFAC(Max_Controls),
     .     UMINFAC(Max_Controls), D, ZERO
      PARAMETER (ZERO = 1E-10)
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / ALLODIAGS / DG_FACET,DG_ISTAT,DG_U_Globals,DG_ET,
     .                     DG_MAXET, DG_IMSG
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
! Get the transformation matrix T which rotates a moment axis perpin-
! dicular to the faces formed by controls U1 and U2.

! find the normal vector to the face formed by controls U1 and U2

      NORMAL(1) = BMAT(2,U1)*BMAT(3,U2) - BMAT(3,U1)*BMAT(2,U2)
      NORMAL(2) = BMAT(3,U1)*BMAT(1,U2) - BMAT(1,U1)*BMAT(3,U2)
      NORMAL(3) = BMAT(1,U1)*BMAT(2,U2) - BMAT(2,U1)*BMAT(1,U2)

! Row one of the transformation matrix is the direction cosine represen-
! tation of the NORMAL vector. Find its length.

      NORMMAG = SQRT(NORMAL(1)**2 + NORMAL(2)**2 + NORMAL(3)**2)

! If NORMMAG is zero then these controls are redundant and we can't
! allocate with them.

      IF (ABS(NORMMAG) .LE. ZERO) THEN
        ISTATUS = 4
        MSG = 'REDUNDANT CONTROLS'
        RETURN
      END IF

1030  CONTINUE       ! These controls form a face in moment space

      DO 1033 K = 1,3
        TR1(K) = NORMAL(K)/NORMMAG    ! Direction cosine vector
1033  CONTINUE

      DO 1035 K = 1,M
        BROW(K) = TR1(1)*BMAT(1,K) + TR1(2)*BMAT(2,K) + TR1(3)*BMAT(3,K)
1035  CONTINUE

! BROW should now have zeros corresponding to U1 and U2. The signs of
! other entries determine if they are a minimum or a maximum on this
```

```
! face. (0 -> minimum deflection, 1 -> maximum deflection, 2 -> some-
! where in between.

      NS = 0
      NN = 0

      DO 1040 K = 1,M
        IF ((K .NE. U1) .AND. (K .NE. U2)) THEN
          IF (ABS(BROW(K)) .LE. ZERO) THEN        ! special controls
            IMINFAC(K) = 0
            IMAXFAC(K) = 0
            NS = NS + 1
            IS(NS) = K
          ELSE
            IF (BROW(K) .GT. 0.0) THEN
              UMINFAC(K) = UMIN(K)
              UMAXFAC(K) = UMAX(K)
              IMINFAC(K) = 0
              IMAXFAC(K) = 1
            ELSE
              UMINFAC(K) = UMAX(K)
              UMAXFAC(K) = UMIN(K)
              IMINFAC(K) = 1
              IMAXFAC(K) = 0
            END IF
            NN = NN + 1
            IN(NN) = K
          END IF
        ELSE
          NN = NN + 1
          IN(NN) = K
          UMINFAC(K) = UMIN(K)
          UMAXFAC(K) = UMIN(K)
          IMINFAC(K) = 2
          IMAXFAC(K) = 2
        END IF

1040  CONTINUE

! We might have exceeded the maximum number of special controls that
! we can handle

      IF (NS .GT. MAXS) THEN
        ISTATUS = 5
        MSG = 'TOO MANY REDUNDANT CONTROLS'
        RETURN
      END IF

! Now if we have any "special" controls (ie. controls other than U1
! and U2 that produce perpindicular faces) we have to guess what they
! might be. We try every possible combination of 0 and 1 for the
```

```
! special controls

      IF (NS .GT. 0) THEN
        J = INT((2**NS)/2 - 1)
      ELSE
        J = 0
      END IF

! Basically we find every possible combination of facet codes for the
! special controls by converting the I counter to binary notation where
! each facet element contains 1 bit. Then based on this facet, we form
! the opposite facet, and call GET_MAT and GET_U as before. If there
! are no special controls, then this loop is only done once.

      DO 1060 I = 0,J
        IF (NS .GT. 0) THEN
          DIV = I
          BIT = NS

! In this loop we convert the counter I into binary form

          DO WHILE (DIV .GT. 0)
            IMAXFAC(IS(BIT)) = JMOD(DIV,2)
            DIV = INT(DIV/2)
            BIT = BIT - 1
          END DO

          DO 1061 K = 1,NS
            IF (IMAXFAC(IS(K)) .EQ. 1) THEN
              UMAXFAC(IS(K)) = UMAX(IS(K))
              IMINFAC(IS(K)) = 0
              UMINFAC(IS(K)) = UMIN(IS(K))
            ELSE
              UMAXFAC(IS(K)) = UMIN(IS(K))
              IMINFAC(IS(K)) = 1
              UMINFAC(IS(K)) = UMAX(IS(K))
            END IF
1061      CONTINUE
        END IF

! Make a matrix whose columns are the vertex vector (ref. from the
! origin), and the two edge vectors (ref. from the vertex).

        CALL GET_MAT (M, MAT_MAX, BMAT, UMAXFAC, U1, U2, UMIN, UMAX)

! Call GET_U to check this facet and allocate controls if possible.

        CALL GET_U (M, UALLO, RSAT, Allocated, ISTATUS, MSG, U1, U2,
     .              MAT_MAX, MOM, UMIN, UMAX, IMAXFAC)

        DO 1100 K = 1,M
```

```
          DG_FACET(K) = IMAXFAC(K)
1100    CONTINUE

        IF (.NOT. Allocated) THEN

!     Check the opposite facet now

          CALL GET_MAT (M, MAT_MIN, BMAT, UMINFAC, U1, U2, UMIN, UMAX)

! Call GET_U to check this facet and allocate controls if possible.

          CALL GET_U (M, UALLO, RSAT, Allocated, ISTATUS, MSG, U1, U2,
     .                MAT_MIN, MOM, UMIN, UMAX, IMINFAC)

          DO 1110 K = 1,M
            DG_FACET(K) = IMINFAC(K)
1110      CONTINUE

        END IF

        IF (Allocated) THEN
          IF (NS .GT. 0) THEN
            ISTATUS = 0
            MSG = 'ALLOCATION SUCCESSFUL: SOME CONTROLS WERE REDUNDANT'
          END IF
          RETURN
        END IF

1060  CONTINUE

! ----------------------------------------------------------------------
!
!     End of GET_FACET
!
! ----------------------------------------------------------------------
      RETURN
      END
C23456789012345678901234567890123456789012345678901234567890123456789012
! ----------------------------------------------------------------------
!
!       Module Name: GET_MAT
!           Called By: GET_FACET
!           Calls To: none
!
! ----------------------------------------------------------------------
      SUBROUTINE GET_MAT (M, MAT, B, U, U1, U2, UMIN, UMAX)
! ----------------------------------------------------------------------
!
!       Function:   This subroutine forms the 3X3 matrix (in moment
!                 space), whos columns consist of the vector to a facet
!                 vertex (ref. from origin) and the vectors of two facet
```

```
!                edges (ref. from vertex)
!
!
! ---------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                          By
!  JUL 09 1996        Created                                 J. Bolling
!
! ---------------------------------------------------------------------
!
!     Glossary Section
!
! ---------------------------------------------------------------------
!                             Global Variables
!
!     Name     |   Type    |           Description
*VARNAME        VARTYPE     VARDESCRIPTION
!
!                             Local Variables
!
!     Name     |   Type    |           Description
*M              INTEGER     Number of controls to allocate
*U1             INTEGER     Number of 1st control defining facet
*U2             INTEGER     Number of 2nd control defining facet
*B(3,M)         REAL*8      Control power matrix
*U(M)           REAL*8      Vector of controls (either min or max def.)
*UMIN(M)        REAL*8      Vector of minimum deflections
*UMAX(M)        REAL*8      Vector of maximum deflections
*MAT(3,3)       REAL*8      Output matrix
! ---------------------------------------------------------------------
      IMPLICIT NONE
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
      INTEGER M, U1, U2
      INTEGER Max_Controls
      PARAMETER (Max_Controls = 20)
      REAL B(3,Max_Controls), U(Max_Controls),
     .     UMIN(Max_Controls), UMAX(Max_Controls), MAT(3,3)

      REAL M0(3), M1(3), M2(3), V1(3), V2(3)
      INTEGER I, J, K
! ---------------------------------------------------------------------
!
!     Initialization Section
!
! ---------------------------------------------------------------------
! Initialize vertices
```

```
      DO 1010 I = 1, 3
        M0(I) = 0.0
        M1(I) = 0.0
        M2(I) = 0.0
1010  CONTINUE
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
! Vertex #0 free controls at (0,0)
      U(U1) = UMIN(U1)
      U(U2) = UMIN(U2)
      DO 1011 I = 1,3
        DO 1012 J = 1,M
          M0(I) = M0(I)+B(I,J)*U(J)
1012    CONTINUE
1011  CONTINUE

! Vertex #1 free controls at (0,1)
      U(U1) = UMIN(U1)
      U(U2) = UMAX(U2)
      DO 1013 I = 1,3
        DO 1014 J = 1,M
          M1(I) = M1(I)+B(I,J)*U(J)
1014    CONTINUE
1013  CONTINUE

! Vertex #2 free controls at (1,0)
      U(U1) = UMAX(U1)
      U(U2) = UMIN(U2)
      DO 1015 I = 1,3
        DO 1016 J = 1,M
          M2(I) = M2(I)+B(I,J)*U(J)
1016    CONTINUE
1015  CONTINUE

! Form matrix. Column 1 is the vector from the origin to vertex #0
! Column 2 is the vector from vertex #0 to vertex #2 (U(U1) varying)
! Column 3 is the vector from vertex #0 to vertex #1 (U(U2) varying)

      DO 1020 J=1,3
        MAT(J,1) = M0(J)
        MAT(J,2) = M2(J)-M0(J)
        MAT(J,3) = M1(J)-M0(J)
1020  CONTINUE

! ---------------------------------------------------------------------
!
!     End Of GET_MAT
!
```

```
! ---------------------------------------------------------------------
      RETURN
      END
```

1.5 Allocating Controls

A. Usage

This module is used once the required facet geometry has been determined to check the facet, and if possible, allocate the controls

A.1 GET_U

Function Prototype, SUBROUTINE

REAL	UALLO(M), SAT, GEOM(3,3), MOM(3), UMIN(M), UMAX(M)
INTEGER	M, ISTATUS, U1, U2, IFAC(M)
LOGICAL	ALLOCATED
CHARACTER*80	MSG

Assign data to GEOM, MOM, UMIN, UMAX, M, IFAC

CALL GET_U(M, UALLO, SAT, ALLOCATED, ISTATUS, MSG, U1, U2, GEOM, MOM, UMIN, UMAX, IFAC)

GET_U performs various tests on the facet defined by GEOM and allocates controls if possible.

Argument Definitions

M	[in]	number of controls to allocate
UALLO(M)	[out]	Allocated control vector (global or delta)
SAT	[out]	amount of rate saturation (of the ΔAMS)
ALLOCATED	[out]	True if controls were allocated successfully, otherwise false
ISTATUS	[out]	Status code of the allocation procedure
MSG	[out]	description of the ISTATUS code
U1	[in]	number of the first facet defining control
U2	[in]	number of the second facet defining control
GEOM(3,3)	[in]	A matrix containing the current facet geometry
MOM(3)	[in]	vector of desired moments (global or delta)
UMIN(M)	[in]	vector of minimum control constraints
UMAX(M)	[in]	vector of maximum control constraints
IFAC(M)	[in]	Facet defining control vector in base 3 notation

B. General Remarks

This module contains many facet checking and special case error handling abilities. Although some of these errors may be impossible. It is always better to be safe than sorry.

C. Functional Description

GET_U first attempts to invert the GEOM matrix through a call to INVMAT3. If the matrix is found to be singular, then error detection begins immediately.

GEOM is singular (check that origin lies on the bounding facet).

First a search is begun for a non-singular 2x2 partition of the GEOM matrix. If none are found then the

facet is degenerate and has co-linear edges. ISTATUS is set to 2 and GET_U returns control to GET_FACET. Otherwise, GET_U attempts to express the vector from the origin to the vertex (M0) in terms of the two edge vectors M1 and M2, (indicating that it does in fact, lie in the plane of the facet). This is done by first solving the 2x2 sub matrix problem for the unknown constants C2 and C3, and then checking that the excluded row of the original 3x3 matrix is also satisfied by these constants. If this does not hold, then some unknown singularity has occurred and GET_U returns with an ISTATUS of 1. The next test involves checking to see if the moment is in the plane of the facet or not. This is done in the same manner that the origin was checked. If the moment does not lie in the plane of the facet, GET_U returns with an ISTATUS of 3. If this test is passed, then the moment lies in the plane of the facet. To check whether or not the moment lies within the facet requires making sure that the two constants C2 and C3 (which are used to find the linear combination of the facet edge vectors) are between 0 and 1. If not, then the moment does not lie within the facet and GET_U returns with an ISTATUS of 3 again. Assuming this test is passed, the controls can be allocated by transforming the base 3 vector IFAC into its related control positions, and scaling the varying controls using the edge vectors (in control space), and the constants C2 and C3 (C1 is 1.0 in this case).

GEOM is non-singular (the typical case).

Under normal circumstances, GET_U solves the system $[C1,C2,C3]^T = [GEOM]^{-1}*MOM$. The saturation in moment space SAT is set to C1. If C1 is 0, (An indication that the moment vector does not intersect the facet) then GET_U returns with ISTATUS set to -1. Otherwise, the moment is scaled to the boundary by dividing the constants C2, and C3 by C1. (C1 should be positive; if not, then GET_U returns with ISTATUS set to 7) Next, C2 and C3 are checked to make sure they lie between zero and one. If so, then ISTATUS is set to 0 and the controls are allocated as before (Except that C1 may lie somewhere between 0 and 1, so that the controls may be scaled back). Otherwise, the current facet is not the correct one and GET_U returns with ISTATUS set to 6.

D. Errors and Restrictions

The descriptions of the different error status codes and where they are defined are described below.

ISTATUS	Set in...	Description
-99	CONALLO	Allocation has not been performed
-1	GET_U	Moment vector is parallel to the current facet
0	GET_U	Controls allocated successfully
1	GET_U	Unknown singularity in geometry (origin not on the boundary)
2	GET_U	Degenerate Facet has co-linear edges
3	GET_U	Origin on the boundary, moment is not
4	GET_FACET	The current pair of face-defining controls are redundant
5	GET_FACET	The number of redundant controls has exceeded the limit
6	GET_U	Normal operation, wrong facet
7	GET_U	Negative saturation C1 < 0

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
```

```
!       Module Name: GET_U
!           Called by: GET_FACET
!           Calls to: INVMAT3,
!
! ----------------------------------------------------------------------
      SUBROUTINE GET_U (M, U, SAT, ALLOCATED, ISTATUS, MSG, U1, U2, MAT,
     .                  MOM, UMIN, UMAX, IFAC)
! ----------------------------------------------------------------------
!
!       Function:   This subroutine checks the facet (Defined by MAT) to
!                 see if we can allocate. If we can, then it solves for
!                 the vector of controls.
!
!
! ----------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                          By
!  APR 10 1997        Changed the ranges on the constants C2, and C3
!                     by a small amount so that round-off errors are
!                     handled better when checking facets and the moment
!                     happens to lie on an edge                     JB
!  JUL 09 1996        Created                                 J. Bolling
!
! ----------------------------------------------------------------------
!
!     Glossary Section
!
! ----------------------------------------------------------------------
!                             Global Variables
!
!     Name     |   Type    |           Description
*VARNAME        VARTYPE     VARDESCRIPTION
!
!                             Local Variables
!
!     Name     |   Type    |           Description
*M              INTEGER     Number of controls to allocate
*U1             INTEGER     Number of 1st control defining facet
*U2             INTEGER     Number of 2nd control defining facet
*U(M)           REAL        Vector of allocated controls
*UMIN(M)        REAL        Vector of minimum deflections
*UMAX(M)        REAL        Vector of maximum deflections
*MAT(3,3)       REAL        Matrix of facet geometry
*SAT            REAL        Saturation level
*ALLOCATED      LOGICAL     Allocated successful flag
*IFAC           INTEGER     Base three facet code
! ----------------------------------------------------------------------
      IMPLICIT NONE
! ----------------------------------------------------------------------
!
```

```
!     Declaration Section
!
! ------------------------------------------------------------------------
      INTEGER M, U1, U2
      INTEGER Max_Controls
      PARAMETER (Max_Controls = 20)
      INTEGER IFAC(Max_Controls), ISTATUS
      REAL U(Max_Controls), MAT(3,3), UMIN(Max_Controls),
     .     UMAX(Max_Controls), MOM(3)
      REAL SAT
      CHARACTER*80 MSG
      LOGICAL ALLOCATED, GOOD

      REAL D, TRANS(3,3), A(2,2), AINV(2,2), C1, C2, C3,
     .     ZERO
      PARAMETER (ZERO = 1E-10)
      INTEGER I,J,K
! ------------------------------------------------------------------------
!
!     Run Section
!
! ------------------------------------------------------------------------
      ALLOCATED = .FALSE.

! Invert MAT to get TRANS

      CALL INVMAT3(TRANS,MAT,D)

! First Check the Determinate of MAT, D

      IF (ABS(D) .LE. ZERO) THEN

! Det is zero ...
! Either the origin is on the facet or the two edges are parallel.
! In the first case, the desired moment may be on the boundary as well,
! in which case, we can allocate. Otherwise, we shouldn't even be here.

! Origin on the boundary? First find a non-singular 2x2 partition

        DO 2007 I=1,2
          DO 2006 J=I+1,3
            A(1,1) = MAT(I,2)
            A(1,2) = MAT(I,3)
            A(2,1) = MAT(J,2)
            A(2,2) = MAT(J,3)
            D = A(1,1)*A(2,2)-A(1,2)*A(2,1)
            IF (ABS(D) .GT. ZERO) GO TO 2008  ! Found one. Keep I and J
2006      CONTINUE
2007    CONTINUE

2008    CONTINUE
```

```
        IF (ABS(D) .LE. ZERO) THEN            ! Couldn't find one
          ISTATUS = 2
          MSG = 'CO-LINEAR EDGES'
          RETURN
        END IF

        IF ((I.EQ.1).AND.(J.EQ.2)) K = 3
        IF ((I.EQ.1).AND.(J.EQ.3)) K = 2
        IF ((I.EQ.2).AND.(J.EQ.3)) K = 1

        AINV(1,1) =  A(2,2)/D
        AINV(2,2) =  A(1,1)/D
        AINV(1,2) = -A(1,2)/D
        AINV(2,1) = -A(2,1)/D

! See if column 1 of MAT lies in the plane of the facet

        C2 = -(AINV(1,1)*MAT(I,1)+AINV(1,2)*MAT(J,1))
        C3 = -(AINV(2,1)*MAT(I,1)+AINV(2,2)*MAT(J,1))
        C1 =  C2*MAT(K,2) + C3*MAT(K,3)

        IF (ABS(MAT(K,1) - C1) .GT. ZERO) THEN  ! some other singularity
          ISTATUS = 1
          MSG = 'SNGLR, ORGN NOT ON BDRY'
          RETURN
        END IF

! Is the moment on the boundary?

        C2 = -(AINV(1,1)*MOM(I)+AINV(1,2)*MOM(J))
        C3 = -(AINV(2,1)*MOM(I)+AINV(2,2)*MOM(J))
        C1 =  C2*MAT(K,2) + C3*MAT(K,3)

        IF (ABS(MOM(K) - C1) .GT. ZERO)  THEN
         ISTATUS = 3
         MSG = 'ORGN ON BDRY, MOM IS NOT'
         RETURN
        END IF

! If we made it this far, then the origin and the moment are on the
! boundary. but does it lie within this facet?

        C1 = 1.0

        GOOD = (     (-0.01 .LE. C2) .AND. (-0.01 .LE. C3)
    .          .AND. (C2 .LE. 1.01) .AND. (C3 .LE. 1.01))

        IF (.NOT. GOOD)  THEN
          ISTATUS = 3
          MSG = 'ORGN ON BDRY, MOM IS NOT'
```

```
          RETURN
        ELSE
          ISTATUS = 0
          MSG = 'ORGN ON BDRY'
          SAT = 1.0
          GO TO 2010              ! We got one. skip the next part
        END IF

      END IF

! We are finished checking the special cases associated with a sing-
! ularity in MAT

2009  CONTINUE                    ! NOW THE NORMAL (?) CASE

! Get C1,C2,C3 as in MOM_IN = C1*VERTEX_0 + C2*EDGE_1 + C3*EDGE_2

      C1 = 0.0
      C2 = 0.0
      C3 = 0.0

      DO 1013 I=1,3
        C1 = C1 + TRANS(1,I)*MOM(I)
        C2 = C2 + TRANS(2,I)*MOM(I)
        C3 = C3 + TRANS(3,I)*MOM(I)
1013  CONTINUE

      SAT = C1                    ! Saturation level

! IF C1 IS ZERO WE SHOULDN'T EVEN BE HERE

      IF (ABS(C1) .LE. ZERO) THEN
        GOOD = .FALSE.
        ISTATUS = -1
        MSG = 'MOMENT || FACET'
        RETURN
      END IF

      GOOD = (C1 .GT. 0.0)        ! So far, so good

      IF (GOOD) THEN

        C2 = C2/C1                ! scale to the boundary
        C3 = C3/C1
        IF (C1 .GT. 1.0) C1 = 1.0

        GOOD = (    (-0.01 .LE. C2) .AND. (-0.01 .LE. C3)
     .         .AND.(C2 .LE. 1.01) .AND. (C3 .LE. 1.01))

        IF (GOOD) THEN
          ISTATUS = 0
```

```
          MSG = 'NORMAL'
        ELSE
          ISTATUS = 6
          MSG = 'ALL OK, JUST WRONG FACET'
        END IF

      ELSE                          ! C1 is negative
        ISTATUS = 7
        MSG = 'NEGATIVE SATURATION'
      END IF

2010  CONTINUE

! If we have a good one, allocate it

      IF (GOOD) THEN

! First allocate and scale the fixed controls

        DO 1014 I=1,M
          IF ((I .NE. U1) .AND. (I .NE. U2)) THEN      ! skip U1 nad U2
            IF (IFAC(I) .EQ. 0) U(I) = UMIN(I)*C1
            IF (IFAC(I) .EQ. 1) U(I) = UMAX(I)*C1
          END IF
1014    CONTINUE

! Now the varying controls

        U(U1) = C1*(UMIN(U1) + C2*(UMAX(U1)-UMIN(U1)))
        U(U2) = C1*(UMIN(U2) + C3*(UMAX(U2)-UMIN(U2)))

        ALLOCATED = .TRUE.

      END IF
! ---------------------------------------------------------------------
!
!     End of GET_U
!
! ---------------------------------------------------------------------
      RETURN
      END
```

1.6 Miscellaneous Subroutines and Functions

A. Usage

This section describes some of the general functions and subroutines that supplement the ConAllo software.

A.1 PINVB4

Function Prototype, SUBROUTINE

```
REAL        BMAT(4,20), PINVB(20,4)
INTEGER     M
```

Assign values to M and BMAT

```
CALL PINVB4(M, BMAT, PINVB)
```

PINVB4 returns the right pseudo inverse of the matrix in BMAT

Argument Definitions

M	[in]	Number of columns in matrix BMAT
BMAT(4,M)	[in]	The 4 x M matrix BMAT
PINVB(M,4)	[out]	The right pseudo inverse of BMAT

A.2 INVMAT3

Function Prototype, SUBROUTINE

```
REAL        MATOUT(3,3),MATIN(3,3),D
```

Assign data to MATIN

```
CALL INVMAT3(MATOUT, MATIN, D)
```

INVMAT3 returns the inverse of MATIN in MATOUT (if it exists), and the determinant of MATIN in D

Argument Definitions

MATOUT(3,3)	[out]	The inverse of MATIN
MATIN(3,3)	[in]	The 3 x 3 matrix to be inverted
D	[out]	The determinant of MATIN

A.3 INVMAT4

Function Prototype, SUBROUTINE

```
REAL        MATOUT(4,4),MATIN(4,4),D
```

Assign data to MATIN

```
CALL INVMAT4(MATOUT, MATIN, D)
```

INVMAT4 returns the inverse of MATIN in MATOUT (if it exists), and the determinant of MATIN in D

Argument Definitions

MATOUT(4,4)	[out]	The inverse of MATIN
MATIN(4,4)	[in]	The 4,4 matrix to be inverted
D	[out]	The determinant of MATIN

A.4 D3

Function Prototype, REAL

REAL	MATIN(3,3)

Assign data to MATIN

DET = D3(MATIN)

D3 returns the determinant of the matrix MATIN

Argument Definitions

MATIN(3,3)	[in]	The 3 x 3 matrix to calculate the determinant of

B. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ----------------------------------------------------------------------
!
!       Module Name: PINVB4
!           Called By: RESTORE_U
!           Calls to: INVMAT4
!
! ----------------------------------------------------------------------
      SUBROUTINE PINVB4(M, BMAT, PINVB)
! ----------------------------------------------------------------------
!
!       Function:    Finds the minimum-norm right inverse P for a 4XM
!                matrix B.
!
!                P = B'*(inv(B*B')) such that BP = I
!
! ----------------------------------------------------------------------
      IMPLICIT NONE
! ----------------------------------------------------------------------
!
!     Declaration Section
!
! ----------------------------------------------------------------------
      INTEGER M,I,J,K,Max_Controls
      PARAMETER (Max_Controls = 20)
      REAL PINVB(Max_Controls,4), BMAT(4,Max_Controls)
      REAL BMATT(Max_Controls,4), BBT(4,4), INVBBT(4,4),
     .     BBTK, PINVBK, D
! ----------------------------------------------------------------------
```

```
!
!     Run Section
!
! ---------------------------------------------------------------------

! B^T

      DO 10 I=1,M
        DO 20 J=1,4
         BMATT(I,J) = BMAT(J,I)
20      CONTINUE
10    CONTINUE

! B*B^T

      DO 30 I=1,4
        DO 40 J=1,4
          BBT(I,J) = 0
          DO 50 K=1,M
            BBTK = BMAT(I,K)*BMATT(K,J)
            BBT(I,J) = BBT(I,J) + BBTK
50        CONTINUE
40      CONTINUE
30    CONTINUE

! [B*B^T]^-1

      CALL INVMAT4(INVBBT,BBT,D)

! (B^T)*[B*B^T]^-1

      DO 60 I=1,M
        DO 70 J=1,4
          PINVB(I,J) = 0
          DO 80 K=1,4
            PINVBK = BMATT(I,K)*INVBBT(K,J)
            PINVB(I,J) = PINVB(I,J) + PINVBK
80        CONTINUE
70      CONTINUE
60    CONTINUE
! ---------------------------------------------------------------------
!
!     End of PINVB4
!
! ---------------------------------------------------------------------
      RETURN
      END
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ---------------------------------------------------------------------
!
!       Module name: INVMAT3
```

```
!           Called by: GET_U
!           Calls to: D3
!
! ----------------------------------------------------------------------
      SUBROUTINE INVMAT3(MATOUT,MATIN,D)
! ----------------------------------------------------------------------
!
!       Function:   Inverts a 3X3 matrix
!
! ----------------------------------------------------------------------
      IMPLICIT NONE
! ----------------------------------------------------------------------
!
!     Declaration Section
!
! ----------------------------------------------------------------------
      REAL MATOUT(3,3), MATIN(3,3), D, D3
      INTEGER I,J

      DO 2 I=1,3
        DO 1 J=1,3
          MATOUT(I,J)=0.0
 1      CONTINUE
 2    CONTINUE

      D = D3(MATIN)

      IF (D.NE.0.0) THEN
        MATOUT(1,1) =  (MATIN(2,2)*MATIN(3,3)-MATIN(2,3)*MATIN(3,2))/D
        MATOUT(1,2) = -(MATIN(1,2)*MATIN(3,3)-MATIN(1,3)*MATIN(3,2))/D
        MATOUT(1,3) =  (MATIN(1,2)*MATIN(2,3)-MATIN(1,3)*MATIN(2,2))/D
        MATOUT(2,1) = -(MATIN(2,1)*MATIN(3,3)-MATIN(2,3)*MATIN(3,1))/D
        MATOUT(2,2) =  (MATIN(1,1)*MATIN(3,3)-MATIN(1,3)*MATIN(3,1))/D
        MATOUT(2,3) = -(MATIN(1,1)*MATIN(2,3)-MATIN(1,3)*MATIN(2,1))/D
        MATOUT(3,1) =  (MATIN(2,1)*MATIN(3,2)-MATIN(2,2)*MATIN(3,1))/D
        MATOUT(3,2) = -(MATIN(1,1)*MATIN(3,2)-MATIN(1,2)*MATIN(3,1))/D
        MATOUT(3,3) =  (MATIN(1,1)*MATIN(2,2)-MATIN(1,2)*MATIN(2,1))/D
      END IF

      RETURN
      END
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ----------------------------------------------------------------------
!
!       Module name: INVMAT4
!           Called by: PINVB4
!           Calls to: D3
!
! ----------------------------------------------------------------------
      SUBROUTINE INVMAT4(MATOUT,MATIN,D)
! ----------------------------------------------------------------------
```

```
!
!       Function:   Inverts a 4x4 matrix
!
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------
      REAL MATOUT(4,4), MATIN(4,4), D
      REAL TEMP3(3,3), TEMP4(4,4), D3
      INTEGER I,J,II,JJ,ROW,COL,ISIGN

      DO 2 I=1,4
        DO 1 J=1,4
          MATOUT(I,J) = 0.0
 1      CONTINUE
 2    CONTINUE

! COFACTORS

      ISIGN = 1
      DO 10 I=1,4
        DO 9 J=1,4

! MAKE 3X3s

          ROW = 1
          DO 8 II=1,3
            IF (I.EQ.II) ROW = ROW+1
            COL = 1
            DO 7 JJ=1,3
              IF (J.EQ.JJ) COL = COL+1
              TEMP3(II,JJ) = MATIN(ROW,COL)
              COL = COL+1
 7          CONTINUE
            ROW = ROW+1
 8        CONTINUE
          TEMP4(I,J) = D3(TEMP3)*ISIGN
          ISIGN = -ISIGN
 9      CONTINUE
        ISIGN = -ISIGN
 10   CONTINUE

! END COFACTORS

! DETERMINANT

      D = 0.0
```

```
      DO 11 J=1,4
        D = D + MATIN(1,J)*TEMP4(1,J)
 11   CONTINUE

      IF (D.EQ.0.0) RETURN

      DO 13 I=1,4
       DO 12 J=1,4
        MATOUT(I,J) = TEMP4(J,I)/D
 12    CONTINUE
 13   CONTINUE

      RETURN
      END
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ---------------------------------------------------------------------
!
!       Module Name: D3
!           Called By: INVMAT3
!           Calls to: none
!
! ---------------------------------------------------------------------
      FUNCTION D3(MATIN)
! ---------------------------------------------------------------------
!
!       Function:   Determinant of a 3X3 matrix
!
! ---------------------------------------------------------------------
      IMPLICIT NONE
      REAL D3, MATIN(3,3)

      D3 = MATIN(1,1)*MATIN(2,2)*MATIN(3,3)
     &   + MATIN(1,2)*MATIN(2,3)*MATIN(3,1)
     &   + MATIN(1,3)*MATIN(2,1)*MATIN(3,2)
     &   - MATIN(1,3)*MATIN(2,2)*MATIN(3,1)
     &   - MATIN(1,2)*MATIN(2,1)*MATIN(3,3)
     &   - MATIN(1,1)*MATIN(2,3)*MATIN(3,2)

      RETURN
      END
```

2. Aircraft-Specific Routines

This section describes the aircraft-specific modules required for gathering control effectiveness data and control position and rate limits. Since these routines must be developed for any aircraft using Control Allocation with Rate Limiting, no pre-defined set of instructions is given. This section merely discusses the function prototypes and calling conventions for each module.

2.1 The Control Effectiveness Lookup Module

A. Usage

A_C$GETUEFF is a dynamically loaded function which is called several times from the CONALLO main executive when building the control effectiveness matrices. While the functional details of this module are left up to the developer to design, it should adhere to the following calling conventions.

A.1 A_C$GETUEFF

Function Prototype, REAL

POINTER	P
INTEGER	IAXIS, IU
REAL	U

Assign values to all arguments.

UEFF = A_C$GETUEFF(P, IAXIS, IU, U)

A_C$GETUEFF returns the control effectiveness on the IAXIS moment or objective axis with respect to the IU control as a function of aircraft states and control positions.

Argument Definitions

P	[in]	Points to the COMMON location of the aircraft state variables.
IAXIS	[in]	Represents the current axis for which to return the control effectiveness.
IU	[in]	Represents the current control index for which to return the control effectiveness.
U	[in]	The current position of the IU control.

B. General Remarks

Since this function is loaded dynamically by CONALLO, it needs the location of any aircraft state variables required to perform the control effectiveness table lookup. For this implementation using the “SimShell1.5” environment, the necessary aircraft state information is stored in the A_CVARS common block, and is passed to A_C$GETUEFF through the POINTER variable P.

The IAXIS parameter indicates the aircraft moment (or objective) axis for which the control effectiveness data should be returned and is defined as follows:

IAXIS Value	Description
1	Return eff. data for the rolling moment axis
2	Return eff. data for the pitching moment axis
3	Return eff. data for the yawing moment axis
4	(Objective axis) Return effectiveness in Drag

If new restoring algorithms are added, then the additional objectives should be given their own IAXIS value.

C. Functional Description

N/A

D. Errors and Restrictions

Recall that CARL allocates a control vector only and has no knowledge of how each entry in the vector corresponds to an actual aircraft control. Therefore, it is important for this lookup routine to keep track of the order of the controls in this vector so that the correct effectiveness data is returned for a given IU argument.

2.2 Control Position and Rate Limits

A. Usage

A_C$GETCSTR is a dynamically loaded subroutine which is called from the CONALLO main executive to get the control position and rate limit vectors. While the functional details of this module are left up to the developer to design, it should adhere to the following calling conventions.

A.1 A_C$GETCSTR

Function Prototype, SUBROUTINE

POINTER	P
INTEGER	M, IUV
REAL	UMAX, UMIN, URMAX, URMIN

Assign values to P, M and the IUV vector.

CALL A_C$GETCSTR(P, M, IUV, UMAX, UMIN, URMAX, URMIN)

A_C$GETCSTR calculates the minimum and maximum position constraints and the respective rate limits in their minimum and maximum directions for the M allocatable controls as a function of aircraft states.

Argument Definitions

P	[in]	Points to the location of the aircraft state variables.
M	[in]	Number of controls to allocate.
IUV(M)	[in]	Contains a list of the aircraft control indices that are to be allocated.
UMAX(M)	[out]	The maximum position limits for the M controls represented by IUV.
UMIN(M)	[out]	The minimum position limits for the M controls represented by IUV.
URMAX(M)	[out]	The deflection rates in the maximum position direction for the M controls represented by IUV.
URMIN(M)	[out]	The deflection rates in the minimum position direction for the M controls represented by IUV.

B. General Remarks

Since this function is loaded dynamically by CONALLO, it needs the location of any aircraft state variables required to perform the control effectiveness table lookup. For this implementation using the “SimShell1.5” environment, the necessary aircraft state information is stored in the A_CVARS common block, and is passed to A_C$GETUEFF through the POINTER variable P.

In contrast to the A_C$GETUEFF function described in Section 2.1, this subroutine accepts and returns vector arguments. The IUV vector then represents the actual control indices used in the allocatable controls vector for the current frame. For instance, if the left stabilator position was stored in the first entry of the aircraft control vector but was the last control allocated by CARL, then IUV(M) would be 1.

C. Functional Description

N/A

D. Errors and Restrictions

The restrictions on A_C$GETUEFF apply to this subroutine as well. Because of the reconfigurable nature of CARL, the order of the controls in the allocatable control vector and the actual control vector may not always be the same. Therefore, careful attention must be given to the IUV vector.

APPENDIX III.

CARL Subroutines for the F-15 ACTIVE

1. Data Initialization For Control Allocation

This section describes the aircraft-specific routines needed to initialize the required Control Allocation with Rate Limiting database and other parameters.

1.1 Required Aircraft Parameters

A. Usage

These routines initialize all of the model-specific control allocation parameters.

A.1 BD_ACSINIT

Function Prototype, SUBROUTINE
COMMON A_CVARS, SIMPARS

This subroutine takes no arguments.

CALL BD_ACSINIT

BD_ACSINIT assigns values to some model specific parameters

Common Definitions

A_CVARS [global] contains all of the aircraft global variables (“SimShell1.5” specific).
SIMVARS [global] contains all of the global simulation variables (“SimShell1.5” specific).

A.2 ACSINIT

Function Prototype, SUBROUTINE
COMMON A_CVARS, SIMPARS

This subroutine takes no arguments.

CALL ACSINIT

ACSINIT performs the remaining initialization procedures not done by BD_ACSINIT.

Common Definitions

A_CVARS [global] contains all of the aircraft global variables (“SimShell1.5” specific).

SIMVARS [global] contains all of the global simulation variables (“SimShell1.5” specific).

B. General Remarks

The purpose of BD_ACSINIT is to assign values to some of the required aircraft and control allocation parameters only and should be regarded as being a BLOCK DATA prototype. Therefore, this subroutine should not contain any equations or calling statements. The ACSINIT subroutine handles all other initialization that may require using equations or calling additional subroutines.

C. Functional Description

BD_ACSINIT initializes the following aircraft parameters via DATA statements:

1.) Mass and geometry parameters; wing area, mean chord, wing span, mass, and reference CG location (in percent mean chord).

2.) Control allocation parameters; the number of aircraft controls, the name of each aircraft control, the controls’ nominal rate and position limits, and actuator time constants for each control surface.

3.) Aircraft states and outputs; the number of states, number of outputs, state names, and output names.

The following parameters are initialized or reset using assignment statements:

1.) Other aircraft parameters; default CG location, name of the aerodynamic database (if used), location of the aircraft symbols database, and control law parameters.

2.) Shell override parameters. Note that all of the SimShell 1.5 flags are initialized to FALSE. This section allows a specific aircraft model to enable a select few.

ACSINIT continues with the initialization process by defining the necessary inertia parameters (as defined in Reference 16), sets the READ_BINARIES flag to TRUE (indicating that the control effectiveness data is saved in binary files), and calls INITUEFF to load the control effectiveness data.

D. Errors and Restrictions

In BD_ACSINIT, the current number of states, outputs, and allocatable controls are 30, 30 and 20 respectively. In ACSINIT, the default value for the READ_BINARIES flag is TRUE. At the time of this writing however, the binary files exist for the Macintosh platform only.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: BD_ACSINIT
!           Called By: A_C$BDINIT
!           Calls to: none
!
! ------------------------------------------------------------------------
      SUBROUTINE BD_ACSINIT
! ------------------------------------------------------------------------
!
!       Function:   Main initialization module for any AirCraft Speci-
!                   fic code. This module initializes the F-15 ACTIVE
!                   V1.0 mass and other specific model parameters
!
!
```

```
! ---------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                          By
!  JUL 03 1996        Created                                  J. Bolling
!  JAN 27 1997        Changed this code module to SUBROUTINE instead
!                     of BLOCK DATR                                 J.B.
!
! ---------------------------------------------------------------------
!
!     Glossary Section
!
! ---------------------------------------------------------------------
!                             Global Variables
!
!     Name     |   Type    |           Description
*Do_Store       LOGICAL     Status of Storage Utility
*Do_Conallo     LOGICAL     Status of Control Allocation
*SaveMoms       LOGICAL     Save moments flag
*SaveCons       LOGICAL     Save controls flag
*S              REAL        Wing Area (ft^2)
*B              REAL        Wing Span (ft)
*CBAR           REAL        Mean Chord (ft)
*XCGR           REAL        Reference CG (% mean chord)
*XCG            REAL        C.G. position (% mean chord)
*MASS           REAL        Aircraft Mass (slugs)
*NCTRLS         INTEGER     Number of configurable controls
*NSTATES        INTEGER     Number of aircraft states
*NOUTS          INTEGER     Number of aircraft outputs
*UNAME(1)       CHARACTER*4 Control name
*XNAME(1)       CHARACTER*4 State name
*YNAME(1)       CHARACTER*4 Output name
*Database       CHARACTER*80Name of current Aero. Database
*SymbolsDB      CHARACTER*80Path to the A/C Symbols.db file
*LAMDAV         REAL        Control Law parameter
*LAMDAW         REAL        Control law parameter
*LAMDAP         REAL        Control law parameter
*LAMDAQ         REAL        Control law parameter
*LAMDAR         REAL        Control law parameter
*URATE0(1)      REAL        Nominal def. rate (deg/s)
*UMIN0(1)       REAL        Def. Limit (deg)
*UMAX0(1)       REAL        Def. limit (deg)
*UTAU(1)        REAL        1st order time constant U(1)
!
! ---------------------------------------------------------------------
      IMPLICIT NONE
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
```

```
      REAL         A_CVARR  (250)
      CHARACTER*4  A_CVARC  (100)
      INTEGER      A_CVARI  ( 20)
      CHARACTER*80 SIMPARC80( 10)
      LOGICAL      SIMPARL  ( 30)

      INTEGER NCTRLS, NSTATES, NOUTS, I
      REAL S, B, CBAR, XCGR, XCG, MASS, LAMDAV, LAMDAW, LAMDAP, LAMDAQ,
     .     LAMDAR
      CHARACTER*4 UNAME(10),XNAME(30),YNAME(30)
      CHARACTER*80 Database,SymbolsDB
      LOGICAL Do_Store, Do_Conallo, SaveMoms, SaveCons
      REAL URATE0(10),UMIN0(10),UMAX0(10),UTAU(10)
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / A_CVARS / A_CVARR,A_CVARI,A_CVARC
      COMMON / SIMPARS / SIMPARL,SIMPARC80

! ---------------------------------------------------------------------
!
!     Equivalence Section
!
! ---------------------------------------------------------------------
      EQUIVALENCE (SIMPARL(7)    , Do_Store       )
      EQUIVALENCE (SIMPARL(3)    , Do_Conallo     )
      EQUIVALENCE (SIMPARL(11)   , SaveMoms       )
      EQUIVALENCE (SIMPARL(10)   , SaveCons       )
      EQUIVALENCE (A_CVARR(71)   , S              )
      EQUIVALENCE (A_CVARR(72)   , B              )
      EQUIVALENCE (A_CVARR(73)   , CBAR           )
      EQUIVALENCE (A_CVARR(70)   , XCGR           )
      EQUIVALENCE (A_CVARR(60)   , XCG            )
      EQUIVALENCE (A_CVARR(69)   , MASS           )
      EQUIVALENCE (A_CVARI(3)    , NCTRLS         )
      EQUIVALENCE (A_CVARI(4)    , NSTATES        )
      EQUIVALENCE (A_CVARI(5)    , NOUTS          )
      EQUIVALENCE (A_CVARC(1)    , UNAME(1)       )
      EQUIVALENCE (A_CVARC(21)   , XNAME(1)       )
      EQUIVALENCE (A_CVARC(51)   , YNAME(1)       )
      EQUIVALENCE (SIMPARC80(1)  , Database       )
      EQUIVALENCE (SIMPARC80(5)  , SymbolsDB      )
      EQUIVALENCE (A_CVARR(28)   , LAMDAV         )
      EQUIVALENCE (A_CVARR(29)   , LAMDAW         )
      EQUIVALENCE (A_CVARR(25)   , LAMDAP         )
      EQUIVALENCE (A_CVARR(26)   , LAMDAQ         )
      EQUIVALENCE (A_CVARR(27)   , LAMDAR         )
      EQUIVALENCE (A_CVARR(84)   , URATE0(1)      )
      EQUIVALENCE (A_CVARR(124)  , UMIN0(1)       )
```

```
      EQUIVALENCE (A_CVARR(104)  , UMAX0(1)       )
      EQUIVALENCE (A_CVARR(144)  , UTAU(1)        )

! ------------------------------------------------------------------------
!
!     Initialization Section
!
! ------------------------------------------------------------------------
! Mass and Geometry
      DATA S    , B    , CBAR , MASS  , XCGR
     .  /  608.0, 42.7 , 15.94, 1264.0, 0.2565                      /

! Control Allocation
      DATA NCTRLS / 9 /
      DATA (UNAME(I),I=1,10)
     .  /  'DHTL', 'DHTR', 'DAL ', 'DAR ', 'DRUD ', 'DCL ' ,'DCR ',
     .     'YNOZ', 'PNOZ', '    '                                   /
      DATA (URATE0(I),I=1,10)
     .  /   45.0,   45.0,   90.0,   90.0,   135.0,   75.0,   75.0,
     .      80.0,   80.0,    0.0                                   /
      DATA (UMIN0(I), I=1,10)
     .  /   -29.0, -29.0,   -20.0,  -20.0,  -30.0,  -35.0,  -35.0,
     .      -20.0, -20.0,     0.0                                 /
      DATA (UMAX0(I), I=1,10)
     .  /    15.0,  15.0,    20.0,   20.0,   30.0,    15.0,   15.0,
     .       20.0,  20.0,     0.0                                 /
      DATA (UTAU(I), I=1,10)
     .  /    10*0.0495                                            /

! States/Outputs
      DATA NSTATES, NOUTS / 23 , 14 /
      DATA (XNAME(I),I=1,30)
     .  /  'VT  ', 'ALFA', 'BETA', 'PHI ', 'THET', 'PSI ', 'P   ',
     .     'Q   ', 'R   ', 'NORT', 'EAST', 'ALT ', 'POW ', 'DHTL',
     .     'DHTR', 'DAL ', 'DAR ', 'DRL ', 'DRR ', 'DCL ', 'DCR ',
     .     'YNOZ', 'PNOZ',     7*'    '                           /
      DATA (YNAME(I), I=1,30)
     .  /  'VT  ', 'ALFA', 'THET', 'Q   ', 'BETA', 'PHI ', 'P   ',
     .     'R   ', 'AN  ', 'ALAT', 'NORT', 'EAST', 'ALT ', 'PSI ',
     .  16*'    '                                                  /

! Other

      XCG = 0.2685
      Database = 'NONE'
      SymbolsDB = ':Models:F15:F-15 Symbols.db'
      LAMDAV = 0.0
      LAMDAW = 0.0
      LAMDAP = 0.0
      LAMDAQ = 0.0
      LAMDAR = 0.0
```

```
! Override the Sim Shell defaults for these variables.

      Do_Store = .TRUE.
      Do_Conallo = .TRUE.
      SaveMoms = .TRUE.
      SaveCons = .TRUE.

! ------------------------------------------------------------------------
!
!     End OF: BD_ACSINIT
!
! ------------------------------------------------------------------------
      RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: ACSINIT
!           Called By: A_C$INIT
!           Calls to: InitUEff
!
! ------------------------------------------------------------------------
      SUBROUTINE ACSINIT
! ------------------------------------------------------------------------
!
!       Function:   Main initialization module for any AirCraft Speci-
!                   fic code. This module initializes the F-15 ACTIVE
!                   V1.0 databases
!
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                          By
!  JUL 03 1996        Created                                  J. Bolling
!
! ------------------------------------------------------------------------
!
!     Glossary Section
!
! ------------------------------------------------------------------------
!                             Global Variables
!
!     Name      |   Type    |           Description
*Do_Conallo     LOGICAL     Status of Control Allocation
*C1             REAL        Inertia Parameter
*C2             REAL        Inertia Parameter
*C3             REAL        Inertia Parameter
*C4             REAL        Inertia Parameter
```

```
*C5             REAL        Inertia Parameter
*C6             REAL        Inertia Parameter
*C7             REAL        Inertia Parameter
*C8             REAL        Inertia Parameter
*C9             REAL        Inertia Parameter
!
! ---------------------------------------------------------------------
      IMPLICIT NONE
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
      REAL         A_CVARR  (250)
      CHARACTER*4  A_CVARC  (100)
      INTEGER      A_CVARI  ( 20)
      CHARACTER*80 SIMPARC80( 10)
      LOGICAL      SIMPARL  ( 30)

      LOGICAL Do_Conallo,Read_Binaries
      REAL C1, C2, C3, C4, C5, C6, C7, C8, C9, GAM, IXX, IYY, IZZ, IXZ
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / A_CVARS / A_CVARR,A_CVARI,A_CVARC
      COMMON / SIMPARS / SIMPARL,SIMPARC80

! ---------------------------------------------------------------------
!
!     Equivalence Section
!
! ---------------------------------------------------------------------
      EQUIVALENCE (SIMPARL(3)    , Do_Conallo     )
      EQUIVALENCE (A_CVARR(74)   , C1             )
      EQUIVALENCE (A_CVARR(75)   , C2             )
      EQUIVALENCE (A_CVARR(76)   , C3             )
      EQUIVALENCE (A_CVARR(77)   , C4             )
      EQUIVALENCE (A_CVARR(78)   , C5             )
      EQUIVALENCE (A_CVARR(79)   , C6             )
      EQUIVALENCE (A_CVARR(80)   , C7             )
      EQUIVALENCE (A_CVARR(81)   , C8             )
      EQUIVALENCE (A_CVARR(82)   , C9             )

! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      DATA IXX, IYY, IZZ ,IXZ / 25919.0, 197590.0, 218138.0, -4894.0 /
```

```
      DATA Read_Binaries / .TRUE. /

! set Inertia parameters for subroutine F

      GAM = IXX*IZZ - IXZ**2
      C1 = ((IYY - IZZ)*IZZ - IXZ**2)/GAM
      C2 = ((IXX - IYY + IZZ)*IXZ)/GAM
      C3 = IZZ/GAM
      C4 = IXZ/GAM
      C5 = (IZZ - IXX)/IYY
      C6 = IXZ/IYY
      C7 = 1.0/IYY
      C8 = (IXX*(IXX - IYY) + IXZ**2)/GAM
      C9 = IXX/GAM

! Load required databases

      IF (Do_Conallo) THEN
        CALL InitUEff(Read_Binaries)
      END IF

! ------------------------------------------------------------------------
!
!     End of ACSINIT
!
! ------------------------------------------------------------------------
      RETURN
      END
```

1.2 Loading The Control Effectiveness Database

A. Usage

These subroutines are used to load the control effectiveness database into memory during the initialization pass. The assumed format is the Affine Data Interpolation technique described in Chapter 2.

A.1 INITUEFF

Function Prototype, SUBROUTINE

LOGICAL	READ_BINARIES
COMMON	S_EFFS, A_EFFS, C_EFFS, R_EFFS, YN_EFFS, PN_EFFS

Assign a value to READ_BINARIES

CALL INITUEFF(READ_BINARIES)

INITUEFF loads the control effectiveness data from either binary files (if READ_BINARIES is TRUE) or from ascii files and stores the data in the ?_EFFS common blocks.

Argument Definitions

READ_BINARIES	[in]	Determines whether or not to load the data from binary files (if they exist).
S_EFFS	[global]	Stabilator effectiveness data.
A_EFFS	[global]	Aileron effectiveness data.
C_EFFS	[global]	Canard effectiveness data.
R_EFFS	[global]	Rudder effectiveness data.
YN_EFFS	[global]	Yaw Nozzle effectiveness data.
PN_EFFS	[global]	Pitch Nozzle effectiveness data.

A.2 LOADMCSDAT

Function Prototype, SUBROUTINE

INTEGER	IU, RN, I1, I2, I3
REAL	RDATA
LOGICAL	READ_BINARIES

Assign values to READ_BINARIES, IU, RN, I1, I2, and I3.

CALL LOADMCSDAT(IU, RN, RDATA, I1, I2, I3, READ_BINARIES)

LOADMCSDAT accesses the record RN in the file specified by unit IU, and returns the I1 by I2 by I3 array of mesh constants in RDATA.

Argument Definitions

IU	[in]	Logical file unit to access.
RN	[in]	Record number to access (used for binary files only).
RDATA	[out]	Mesh constant I1 by I2 by I3 data array.
I1	[in]	Dimensional size of RDATA.
I2	[in]	Dimensional size of RDATA.

I3	[in]	Dimensional size of RDATA.
READ_BINARIES	[in]	Determines whether or not to load the data from binary files.

B. General Remarks

LOADMCSDAT performs all of the file IO and is called multiple times by INITUEFF to load the appropriate data. INITUEFF simply checks whether or not the binary data files exist, sets the READ_BINARIES flag accordingly, and provides the necessary arguments to LOADMCSDAT for each array of mesh constants.

C. Functional Description

If the READ_BINARIES argument is TRUE when INITUEFF is called, then it first checks whether or not a binary file exists. If so, then it instructs LOADMCSDAT to use the binary formatted files. If not, then it assumes no binary files exist and calls LOADMCSDAT with READ_BINARIES set to FALSE.

LOADMCSDAT reads files according to the READ_BINARIES flag. When its value is TRUE, LOADMCSDAT reads files via direct access mode in which each row of the mesh constant matrices is stored as a separate record. When READ_BINARIES is FALSE, LOADMCSDAT reads the data according to the format used by the Matlab utility “mat2ascii” described in Appendix I.

D. Errors and Restrictions

The stored data arrays begin with an index of 0. Therefore, the dimensional parameters I1, I2, and I3 must be 1 less than the table sizes. For example, a 3 by 2 by 6 affine data table would be represented by I1, I2, and I3 as 2,1 and 5.

INITUEFF only checks for the existence of one of the binary files when READ_BINARIES is TRUE. If one or more of these files is missing, then an IO error may occur when an attempt to open the missing file is made.

The data file format for ascii mode is restricted to comply with the “mat2ascii” output with ISF77 set to 1 (see Appendix I). In addition, comment lines may be used in these files as long as the first column contains either a lowercase or uppercase “C”.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: InitUEff
!       Called By: ACSINIT
!       Calls to: LOADMCSDAT
!
! ------------------------------------------------------------------------
      SUBROUTINE InitUEff(BINARY)
! ------------------------------------------------------------------------
!
!       Function:   Initializes the control effectiveness data.
!
!
! ------------------------------------------------------------------------
!
```

```
!       Modifications:
!     Date                        Purpose                          By
!  JUL 23 1996        Created                                      J.B.
!
! ------------------------------------------------------------------------
!
!     Glossary Section
!
! ------------------------------------------------------------------------
!                             Global Variables
!
!     Name     |   Type    |           Description
*VARNAME        VARTYPE     VARDESCRIPTION
!
!                             Local Variables
!
!     Name     |   Type    |           Description
*VARNAME        VARTYPE     VARDESCRIPTION
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------

      INCLUDE 'UEFF_Commons.inc'

      INTEGER I,J, IU, RN
      LOGICAL BINARY,DOBINARY,havebin
! -------------------------------Local------------------------------------
      CHARACTER*17 DataPath
      PARAMETER (DataPath = ':Models:F15:DATA:')
! ------------------------------------------------------------------------
!
!     Common Section
!
! ------------------------------------------------------------------------

! ------------------------------------------------------------------------
!
!     Equivalence Section
!
! ------------------------------------------------------------------------

! ------------------------------------------------------------------------
!
!     Initialization Section
!
! ------------------------------------------------------------------------
!       load the control power data
```

```
 IF (BINARY) THEN
   INQUIRE (FILE = DataPath//'MADS_MCS.bin', EXIST = havebin)
   IF (havebin) THEN
     DOBINARY = .TRUE.
   ELSE
     DOBINARY = .FALSE.
   END IF
 ELSE
   DOBINARY = .FALSE.
 END IF

 IU = 1
 IF (DOBINARY) THEN
   OPEN (UNIT = IU, ACCESS = 'DIRECT',
.        FILE = Datapath//'MADS_MCS.bin', STATUS = 'OLD')
 ELSE
   OPEN (UNIT = IU, FILE = Datapath//'MADS_MCS.dat',
.        STATUS = 'OLD')
 END IF

 RN = 0
 CALL LOADMCSDAT (IU, RN, SC1_MAD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SC1_MA, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SC1_MD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SC1_AD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SC1_M, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SC1_A, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SC1_D, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SC10, SMI, SDI, SAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, SCM_MAD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCM_MA, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCM_MD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCM_AD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCM_M, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCM_A, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCM_D, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCM0, SMI, SDI, SAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, SCN_MAD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCN_MA, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCN_MD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCN_AD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCN_M, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCN_A, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCN_D, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCN0, SMI, SDI, SAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, SCD_MAD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCD_MA, SMI, SDI, SAI, DOBINARY)
```

```
 CALL LOADMCSDAT (IU, RN, SCD_MD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCD_AD, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCD_M, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCD_A, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCD_D, SMI, SDI, SAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, SCD0, SMI, SDI, SAI, DOBINARY)

 CLOSE(IU)

 IF (DOBINARY) THEN
   OPEN (UNIT = IU, ACCESS = 'DIRECT',
.        FILE = Datapath//'MADA_MCS.bin', STATUS = 'OLD')
 ELSE
   OPEN (UNIT = IU, FILE = Datapath//'MADA_MCS.dat',
.        STATUS = 'OLD')
 END IF

 RN = 0
 CALL LOADMCSDAT (IU, RN, AC1_MAD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, AC1_MA, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, AC1_MD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, AC1_AD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, AC1_M, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, AC1_A, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, AC1_D, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, AC10, AMI, ADI, AAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, ACM_MAD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACM_MA, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACM_MD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACM_AD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACM_M, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACM_A, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACM_D, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACM0, AMI, ADI, AAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, ACN_MAD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACN_MA, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACN_MD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACN_AD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACN_M, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACN_A, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACN_D, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACN0, AMI, ADI, AAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, ACD_MAD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACD_MA, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACD_MD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACD_AD, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACD_M, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACD_A, AMI, ADI, AAI, DOBINARY)
```

```
 CALL LOADMCSDAT (IU, RN, ACD_D, AMI, ADI, AAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, ACD0, AMI, ADI, AAI, DOBINARY)

 CLOSE(IU)
 IF (DOBINARY) THEN
   OPEN (UNIT = IU, ACCESS = 'DIRECT',
.        FILE = Datapath//'MADR_MCS.bin', STATUS = 'OLD')
 ELSE
   OPEN (UNIT = IU, FILE = Datapath//'MADR_MCS.dat',
.        STATUS = 'OLD')
 END IF

 RN = 0
 CALL LOADMCSDAT (IU, RN, RC1_MAD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RC1_MA, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RC1_MD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RC1_AD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RC1_M, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RC1_A, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RC1_D, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RC10, RMI, RDI, RAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, RCM_MAD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCM_MA, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCM_MD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCM_AD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCM_M, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCM_A, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCM_D, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCM0, RMI, RDI, RAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, RCN_MAD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCN_MA, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCN_MD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCN_AD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCN_M, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCN_A, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCN_D, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCN0, RMI, RDI, RAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, RCD_MAD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCD_MA, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCD_MD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCD_AD, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCD_M, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCD_A, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCD_D, RMI, RDI, RAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, RCD0, RMI, RDI, RAI, DOBINARY)

 CLOSE(IU)
```

```
 IF (DOBINARY) THEN
   OPEN (UNIT = IU, ACCESS = 'DIRECT',
.        FILE = Datapath//'MADC_MCS.bin', STATUS = 'OLD')
 ELSE
   OPEN (UNIT = IU, FILE = Datapath//'MADC_MCS.dat',
.        STATUS = 'OLD')
 END IF

 RN = 0
 CALL LOADMCSDAT (IU, RN, CC1_MAD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CC1_MA, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CC1_MD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CC1_AD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CC1_M, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CC1_A, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CC1_D, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CC10, CMI, CDI, CAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, CCM_MAD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCM_MA, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCM_MD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCM_AD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCM_M, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCM_A, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCM_D, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCM0, CMI, CDI, CAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, CCN_MAD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCN_MA, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCN_MD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCN_AD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCN_M, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCN_A, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCN_D, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCN0, CMI, CDI, CAI, DOBINARY)

 CALL LOADMCSDAT (IU, RN, CCD_MAD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCD_MA, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCD_MD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCD_AD, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCD_M, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCD_A, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCD_D, CMI, CDI, CAI, DOBINARY)
 CALL LOADMCSDAT (IU, RN, CCD0, CMI, CDI, CAI, DOBINARY)

 CLOSE(IU)

 IF (DOBINARY) THEN
   OPEN (UNIT = IU, ACCESS = 'DIRECT',
.        FILE = Datapath//'MADYN_MCS.bin',
.        STATUS = 'OLD')
```

```
      ELSE
        OPEN (UNIT = IU, FILE = Datapath//'MADYN_MCS.dat',
     .        STATUS = 'OLD')
      END IF

      RN = 0
      CALL LOADMCSDAT (IU, RN, YNCN_MAD, 48, YNDI, YNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, YNCN_MA, 48, YNDI, YNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, YNCN_MD, 48, YNDI, YNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, YNCN_AD, 48, YNDI, YNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, YNCN_M, 48, YNDI, YNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, YNCN_A, 48, YNDI, YNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, YNCN_D, 48, YNDI, YNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, YNCN0, 48, YNDI, YNAI, DOBINARY)

      CLOSE(IU)

      IF (DOBINARY) THEN
        OPEN (UNIT = IU, ACCESS = 'DIRECT',
     .        FILE = Datapath//'MADPN_MCS.bin',
     .        STATUS = 'OLD')
      ELSE
        OPEN (UNIT = IU, FILE = Datapath//'MADPN_MCS.dat',
     .        STATUS = 'OLD')
      END IF

      RN = 0
      CALL LOADMCSDAT (IU, RN, PNCM_MAD, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCM_MA, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCM_MD, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCM_AD, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCM_M, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCM_A, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCM_D, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCM0, 48, PNDI, PNAI, DOBINARY)

      CALL LOADMCSDAT (IU, RN, PNCD_MAD, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCD_MA, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCD_MD, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCD_AD, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCD_M, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCD_A, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCD_D, 48, PNDI, PNAI, DOBINARY)
      CALL LOADMCSDAT (IU, RN, PNCD0, 48, PNDI, PNAI, DOBINARY)

      CLOSE(IU)
! ---------------------------------------------------------------------
!
!     End of InitUEff
!
! ---------------------------------------------------------------------
```

```
      RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: LOADMCSDAT
!       Called By: INITUEFF
!       Calls to: none
!
! ------------------------------------------------------------------------
      SUBROUTINE LOADMCSDAT (IU, RN, RDATA, I1, I2, I3, BINARY)
! ------------------------------------------------------------------------
!
!       Function:   Loads the MCS data files for control allocation into
!                memory
!
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                          By
!  JUL 23 1996        Created                                      J.B.
!
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------
      INTEGER IU, I1, I2, I3, J, K, L, RN
      LOGICAL BINARY
      CHARACTER*1 C1
      REAL RDATA(0:I1,0:I2,0:I3)

! ------------------------------------------------------------------------
!
!     Run Section
!
! ------------------------------------------------------------------------
      IF (BINARY) THEN

        DO 1010 J = 0,I1
          DO 1020 K = 0,I2
            RN = RN + 1
            READ(IU,rec = RN) (RDATA(J,K,L),L = 0,I3)
1020      CONTINUE
1010    CONTINUE

      END IF
```

```
      IF (.NOT. BINARY) THEN

        DO 1030 J = 0,I1
          DO 1040 K = 0,I2
            READ(IU,110) C1
            DO WHILE (C1 .EQ. 'c' .OR. C1 .EQ. 'C')
              READ(IU,110) C1
            END DO
            BACKSPACE (IU)
            READ (IU,120) (RDATA(J,K,L),L = 0,I3)
1040      CONTINUE
1030    CONTINUE

      END IF

110   FORMAT(A1)
120   FORMAT(4(2x,E13.6),/,4(2x,E13.6),/,2(2x,E13.6))
! ----------------------------------------------------------------------
!
!     End of LOADMCSDAT
!
! ----------------------------------------------------------------------
      RETURN
      END

C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ----------------------------------------------------------------------
!
!       File Name: UEFF_Commons.inc
!
! ----------------------------------------------------------------------
      REAL SC1_MAD(0:8,0:8,0:9),SC1_MA(0:8,0:8,0:9),SC1_MD(0:8,0:8,0:9),
     .     SC1_AD(0:8,0:8,0:9) ,SC1_M(0:8,0:8,0:9) ,SC1_A(0:8,0:8,0:9) ,
     .     SC1_D(0:8,0:8,0:9)  ,SC10(0:8,0:8,0:9)

      REAL SCM_MAD(0:8,0:8,0:9),SCM_MA(0:8,0:8,0:9),SCM_MD(0:8,0:8,0:9),
     .     SCM_AD(0:8,0:8,0:9) ,SCM_M(0:8,0:8,0:9) ,SCM_A(0:8,0:8,0:9) ,
     .     SCM_D(0:8,0:8,0:9)  ,SCM0(0:8,0:8,0:9)

      REAL SCN_MAD(0:8,0:8,0:9),SCN_MA(0:8,0:8,0:9),SCN_MD(0:8,0:8,0:9),
     .     SCN_AD(0:8,0:8,0:9) ,SCN_M(0:8,0:8,0:9) ,SCN_A(0:8,0:8,0:9) ,
     .     SCN_D(0:8,0:8,0:9)  ,SCN0(0:8,0:8,0:9)

      REAL SCD_MAD(0:8,0:8,0:9),SCD_MA(0:8,0:8,0:9),SCD_MD(0:8,0:8,0:9),
     .     SCD_AD(0:8,0:8,0:9) ,SCD_M(0:8,0:8,0:9) ,SCD_A(0:8,0:8,0:9) ,
     .     SCD_D(0:8,0:8,0:9)  ,SCD0(0:8,0:8,0:9)

      REAL    SMMIN, SAMIN, SDMIN, SMINC, SAINC, SDINC
      INTEGER SMI  , SAI  , SDI
      DATA    SMMIN, SAMIN, SDMIN, SMINC, SAINC, SDINC
```

```
.      /  0.2 , -10.0, -30.0,  0.2 ,  5.0 ,  5.0  /
 DATA    SMI  , SAI  , SDI
.      /  8   ,  9   ,  8  /

 COMMON /S_EFFS/ SC1_MAD, SCM_MAD, SCN_MAD, SCD_MAD,
.                SC1_MA , SCM_MA , SCN_MA , SCD_MA ,
.                SC1_AD , SCM_AD , SCN_AD , SCD_AD ,
.                SC1_MD , SCM_MD , SCN_MD , SCD_MD ,
.                SC1_M  , SCM_M  , SCN_M  , SCD_M  ,
.                SC1_A  , SCM_A  , SCN_A  , SCD_A  ,
.                SC1_D  , SCM_D  , SCN_D  , SCD_D  ,
.                SC10   , SCM0   , SCN0   , SCD0   ,
.                SMMIN  , SAMIN  , SDMIN  , SMINC  , SAINC , SDINC,
.                SMI    , SAI    , SDI

 REAL AC1_MAD(0:8,0:7,0:9),AC1_MA(0:8,0:7,0:9),AC1_MD(0:8,0:7,0:9),
.     AC1_AD(0:8,0:7,0:9) ,AC1_M(0:8,0:7,0:9) ,AC1_A(0:8,0:7,0:9) ,
.     AC1_D(0:8,0:7,0:9)  ,AC10(0:8,0:7,0:9)

 REAL ACM_MAD(0:8,0:7,0:9),ACM_MA(0:8,0:7,0:9),ACM_MD(0:8,0:7,0:9),
.     ACM_AD(0:8,0:7,0:9) ,ACM_M(0:8,0:7,0:9) ,ACM_A(0:8,0:7,0:9) ,
.     ACM_D(0:8,0:7,0:9)  ,ACM0(0:8,0:7,0:9)

 REAL ACN_MAD(0:8,0:7,0:9),ACN_MA(0:8,0:7,0:9),ACN_MD(0:8,0:7,0:9),
.     ACN_AD(0:8,0:7,0:9) ,ACN_M(0:8,0:7,0:9) ,ACN_A(0:8,0:7,0:9) ,
.     ACN_D(0:8,0:7,0:9)  ,ACN0(0:8,0:7,0:9)

 REAL ACD_MAD(0:8,0:7,0:9),ACD_MA(0:8,0:7,0:9),ACD_MD(0:8,0:7,0:9),
.     ACD_AD(0:8,0:7,0:9) ,ACD_M(0:8,0:7,0:9) ,ACD_A(0:8,0:7,0:9) ,
.     ACD_D(0:8,0:7,0:9)  ,ACD0(0:8,0:7,0:9)

 REAL    AMMIN, AAMIN, ADMIN, AMINC, AAINC, ADINC
 INTEGER AMI  , AAI  , ADI
 DATA    AMMIN, AAMIN, ADMIN, AMINC, AAINC, ADINC
.      /  0.2 , -10.0, -20.0,  0.2 ,  5.0 ,  5.0  /
 DATA    AMI  , AAI  , ADI
.      /  8   ,  9   ,  7  /

 COMMON /A_EFFS/ AC1_MAD, ACM_MAD, ACN_MAD, ACD_MAD,
.                AC1_MA , ACM_MA , ACN_MA , ACD_MA ,
.                AC1_AD , ACM_AD , ACN_AD , ACD_AD ,
.                AC1_MD , ACM_MD , ACN_MD , ACD_MD ,
.                AC1_M  , ACM_M  , ACN_M  , ACD_M  ,
.                AC1_A  , ACM_A  , ACN_A  , ACD_A  ,
.                AC1_D  , ACM_D  , ACN_D  , ACD_D  ,
.                AC10   , ACM0   , ACN0   , ACD0   ,
.                AMMIN  , AAMIN  , ADMIN  , AMINC  , AAINC , ADINC,
.                AMI    , AAI    , ADI

 REAL RC1_MAD(0:8,0:11,0:9),   RC1_MA(0:8,0:11,0:9),
.     RC1_MD(0:8,0:11,0:9) ,   RC1_AD(0:8,0:11,0:9),
```

```
.     RC1_M(0:8,0:11,0:9)  ,   RC1_A(0:8,0:11,0:9) ,
.     RC1_D(0:8,0:11,0:9)  ,   RC10(0:8,0:11,0:9)

 REAL RCM_MAD(0:8,0:11,0:9),   RCM_MA(0:8,0:11,0:9),
.     RCM_MD(0:8,0:11,0:9) ,   RCM_AD(0:8,0:11,0:9),
.     RCM_M(0:8,0:11,0:9)  ,   RCM_A(0:8,0:11,0:9) ,
.     RCM_D(0:8,0:11,0:9)  ,   RCM0(0:8,0:11,0:9)

 REAL RCN_MAD(0:8,0:11,0:9),   RCN_MA(0:8,0:11,0:9),
.     RCN_MD(0:8,0:11,0:9) ,   RCN_AD(0:8,0:11,0:9),
.     RCN_M(0:8,0:11,0:9)  ,   RCN_A(0:8,0:11,0:9) ,
.     RCN_D(0:8,0:11,0:9)  ,   RCN0(0:8,0:11,0:9)

 REAL RCD_MAD(0:8,0:11,0:9),   RCD_MA(0:8,0:11,0:9),
.     RCD_MD(0:8,0:11,0:9) ,   RCD_AD(0:8,0:11,0:9),
.     RCD_M(0:8,0:11,0:9)  ,   RCD_A(0:8,0:11,0:9) ,
.     RCD_D(0:8,0:11,0:9)  ,   RCD0(0:8,0:11,0:9)

 REAL    RMMIN, RAMIN, RDMIN, RMINC, RAINC, RDINC
 INTEGER RMI  , RAI  , RDI
 DATA    RMMIN, RAMIN, RDMIN, RMINC, RAINC, RDINC
.      /  0.2 , -10.0, -30.0,  0.2 ,  5.0 ,  5.0  /
 DATA    RMI  , RAI  , RDI
.      /  8   ,  9   , 11  /

 COMMON /R_EFFS/ RC1_MAD, RCM_MAD, RCN_MAD, RCD_MAD,
.                RC1_MA , RCM_MA , RCN_MA , RCD_MA ,
.                RC1_AD , RCM_AD , RCN_AD , RCD_AD ,
.                RC1_MD , RCM_MD , RCN_MD , RCD_MD ,
.                RC1_M  , RCM_M  , RCN_M  , RCD_M  ,
.                RC1_A  , RCM_A  , RCN_A  , RCD_A  ,
.                RC1_D  , RCM_D  , RCN_D  , RCD_D  ,
.                RC10   , RCM0   , RCN0   , RCD0   ,
.                RMMIN  , RAMIN  , RDMIN  , RMINC  , RAINC , RDINC,
.                RMI    , RAI    , RDI

 REAL CC1_MAD(0:8,0:9,0:9),   CC1_MA(0:8,0:9,0:9),
.     CC1_MD(0:8,0:9,0:9) ,   CC1_AD(0:8,0:9,0:9),
.     CC1_M(0:8,0:9,0:9)  ,   CC1_A(0:8,0:9,0:9) ,
.     CC1_D(0:8,0:9,0:9)  ,   CC10(0:8,0:9,0:9)

 REAL CCM_MAD(0:8,0:9,0:9),   CCM_MA(0:8,0:9,0:9),
.     CCM_MD(0:8,0:9,0:9) ,   CCM_AD(0:8,0:9,0:9),
.     CCM_M(0:8,0:9,0:9)  ,   CCM_A(0:8,0:9,0:9) ,
.     CCM_D(0:8,0:9,0:9)  ,   CCM0(0:8,0:9,0:9)

 REAL CCN_MAD(0:8,0:9,0:9),   CCN_MA(0:8,0:9,0:9),
.     CCN_MD(0:8,0:9,0:9) ,   CCN_AD(0:8,0:9,0:9),
.     CCN_M(0:8,0:9,0:9)  ,   CCN_A(0:8,0:9,0:9) ,
.     CCN_D(0:8,0:9,0:9)  ,   CCN0(0:8,0:9,0:9)
```

```
 REAL CCD_MAD(0:8,0:9,0:9),   CCD_MA(0:8,0:9,0:9),
.     CCD_MD(0:8,0:9,0:9) ,   CCD_AD(0:8,0:9,0:9),
.     CCD_M(0:8,0:9,0:9)  ,   CCD_A(0:8,0:9,0:9) ,
.     CCD_D(0:8,0:9,0:9)  ,   CCD0(0:8,0:9,0:9)

 REAL    CMMIN, CAMIN, CDMIN, CMINC, CAINC, CDINC
 INTEGER CMI  , CAI  , CDI
 DATA    CMMIN, CAMIN, CDMIN, CMINC, CAINC, CDINC
.      /  0.2 , -10.0, -35.0,  0.2 ,  5.0 ,  5.0  /
 DATA    CMI  , CAI  , CDI
.      /  8   ,  9   ,  9  /

 COMMON /C_EFFS/ CC1_MAD, CCM_MAD, CCN_MAD, CCD_MAD,
.                CC1_MA , CCM_MA , CCN_MA , CCD_MA ,
.                CC1_AD , CCM_AD , CCN_AD , CCD_AD ,
.                CC1_MD , CCM_MD , CCN_MD , CCD_MD ,
.                CC1_M  , CCM_M  , CCN_M  , CCD_M  ,
.                CC1_A  , CCM_A  , CCN_A  , CCD_A  ,
.                CC1_D  , CCM_D  , CCN_D  , CCD_D  ,
.                CC10   , CCM0   , CCN0   , CCD0   ,
.                CMMIN  , CAMIN  , CDMIN  , CMINC  , CAINC , CDINC,
.                CMI    , CAI    , CDI


 REAL YNCN_MAD(0:48,0:6,0:7),   YNCN_MA(0:48,0:6,0:7),
.     YNCN_MD(0:48,0:6,0:7) ,   YNCN_AD(0:48,0:6,0:7),
.     YNCN_M(0:48,0:6,0:7)  ,   YNCN_A(0:48,0:6,0:7) ,
.     YNCN_D(0:48,0:6,0:7)  ,   YNCN0(0:48,0:6,0:7)


 REAL    YNMMIN, YNAMIN, YNDMIN, YNMINC, YNAINC, YNDINC
 INTEGER YNMI  , YNAI  , YNDI
 DATA    YNMMIN, YNAMIN, YNDMIN, YNMINC, YNAINC, YNDINC
.      /  0.2 , -10.0, -35.0,  0.2 ,  5.0 ,  5.0  /
 DATA    YNMI  , YNAI  , YNDI
.      /  6   ,  6   ,  7  /

 COMMON /YN_EFFS/ YNCN_MAD, YNCN_MA , YNCN_AD , YNCN_MD ,
.                 YNCN_M  , YNCN_A  , YNCN_D  , YNCN0   ,
.                 YNMMIN  , YNAMIN  , YNDMIN  , YNMINC  ,
.                 YNAINC  , YNDINC  , YNMI    , YNAI    ,
.                 YNDI

 REAL PNCM_MAD(0:48,0:6,0:7),   PNCM_MA(0:48,0:6,0:7),
.     PNCM_MD(0:48,0:6,0:7) ,   PNCM_AD(0:48,0:6,0:7),
.     PNCM_M(0:48,0:6,0:7)  ,   PNCM_A(0:48,0:6,0:7) ,
.     PNCM_D(0:48,0:6,0:7)  ,   PNCM0(0:48,0:6,0:7)


 REAL PNCD_MAD(0:48,0:6,0:7),   PNCD_MA(0:48,0:6,0:7),
.     PNCD_MD(0:48,0:6,0:7) ,   PNCD_AD(0:48,0:6,0:7),
```

```
     .     PNCD_M(0:48,0:6,0:7)  ,   PNCD_A(0:48,0:6,0:7) ,
     .     PNCD_D(0:48,0:6,0:7)  ,   PNCD0(0:48,0:6,0:7)

      REAL    PNMMIN, PNAMIN, PNDMIN, PNMINC, PNAINC, PNDINC
      INTEGER PNMI  , PNAI  , PNDI
      DATA    PNMMIN, PNAMIN, PNDMIN, PNMINC, PNAINC, PNDINC
     .      /  0.2 , -10.0, -35.0,  0.2 ,  5.0 ,  5.0  /
      DATA    PNMI  , PNAI  , PNDI
     .      /  6   ,  6   ,  7  /

      COMMON /PN_EFFS/ PNCM_MAD, PNCM_MA , PNCM_AD , PNCM_MD ,
     .                 PNCM_M  , PNCM_A  , PNCM_D  , PNCM0   ,
     .                 PNCD_MAD, PNCD_MA , PNCD_AD , PNCD_MD ,
     .                 PNCD_M  , PNCD_A  , PNCD_D  , PNCD0   ,
     .                 PNMMIN  , PNAMIN  , PNDMIN  , PNMINC  ,
     .                 PNAINC  , PNDINC  , PNMI    , PNAI    ,
     .                 PNDI
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
```

2. Subroutines Called By CARL

This section describes the GETUEFF and GETCSTR routines specific to the F-15 ACTIVE implementation.

2.1 Control Effectiveness Lookups

A. Usage

This function is used to gather control effectiveness for a specified control and axis using the Affine Data Interpolation technique.

A.1 GETUEFF

Function Prototype, REAL

INTEGER	IAXIS, ICONTROL
REAL	DU
COMMON	A_CVARS, S_EFFS, A_EFFS, C_EFFS, R_EFFS, YN_EFFS, PN_EFFS

Assign values to all arguments

UEFF = GETUEFF(IAXIS, ICONTROL, DU)

GETUEFF returns the control effectiveness for the control surface represented by ICONTROL, on the axis represented by IAXIS, based on Mach number, angle of attack, control deflection DU, and nozzle pressure ratio.

Argument Definitions

IAXIS	[in]	Represents the current axis to return control effectiveness data in.
ICONTROL	[in]	Represents the control for which to gather control effectiveness data.
DU	[in]	The current control surface deflection.
A_CVARS	[global]	Contains all of the aircraft global variables (“SimShell1.5” specific).
S_EFFS	[global]	Stabilator effectiveness data.
A_EFFS	[global]	Aileron effectiveness data.
C_EFFS	[global]	Canard effectiveness data.
R_EFFS	[global]	Rudder effectiveness data.
YN_EFFS	[global]	Yaw Nozzle effectiveness data.
PN_EFFS	[global]	Pitch Nozzle effectiveness data.

B. General Remarks

The IAXIS parameters are defined as follows:

1 Rolling moment axis
2 Pitching moment axis
3 Yawing moment axis
4 Drag axis

The order of the allocatable controls is generally aircraft dependent. They are arranged in the following order for this implementation:

1 Left Stabilator
2 Right Stabilator
3 Left Aileron
4 Right Aileron

5 Rudder (symmetric)
6 Left Canard
7 Right Canard
8 Yaw Nozzle
9 Pitch Nozzle

To minimize the size of the control effectiveness tables however, data is stored for the left controls only. For right control surfaces, a multiplication factor of -1 is applied to the rolling moment and yawing moment axes.

C. Functional Description

Depending of the control index, GETUEFF first finds the location in the appropriate Affine Data Interpolation table based on the current Mach number, angle of attack, and control position. It then finds the 8 mesh constants for that table block and uses them in the 3-D affine data equation presented in Chapter 2 along with the appropriate multiplication factor as described above.

For the Yaw Nozzle and Pitch Nozzle controls, the previously mentioned mesh constants are calculated for the two “bracketing” nozzle pressure ratios to give two results. A linear interpolation is then performed between the two.

D. Errors and Restrictions

If the INITUEFF subroutine is not called at least once before this function is invoked, either incorrect data will be returned or an error will occur. In addition, if the IAXIS or ICONTROL parameters are out of the specified bounds, GETUEFF will return 0.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: GetUEff
!           Called By: CONALLO (or A_C$GETUEFF)
!           Calls to: none
!
! ------------------------------------------------------------------------
      FUNCTION GetUEFF (MAXIS,CONTROL,DU)
! ------------------------------------------------------------------------
!
!       Function: This FUNCTION returns a given control effectiveness on
!                 a given direction in objective space (Cl,Cm,Cn,CD) as a
!                 function of MACH, ALPHA, and NPR and current Control
!                 position.
!
!       Nomenclature: The moment axes Cl Cm Cn are referred to as 1 2
!                     and 3 respectively. The CD axis is defined as axis
!                     4. Controls are defined as follows:
!            DHTL -> U(1)
!            DHTR -> U(2)
!            DAL  -> U(3)
!            DAR  -> U(4)
!            DRUDD      -> U(5)
!            DCANL      -> U(6)
```

```
!            DCANR  -> U(7)
!            PNOZ -> U(8)
!            YNOZ -> U(9)
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                          By
!  JUL 23 1996        Created                                        J.B.
!
! ------------------------------------------------------------------------
!
!     Glossary Section
!
! ------------------------------------------------------------------------
!                             Global Variables
!
!     Name     |   Type    |           Description
*ALPHA          REAL        Angle of attack (degrees)
*MACH           REAL        Mach number (ND)
*NPR            REAL        Nozzle Pressure Ratio
!
!                             Local Variables
!
!     Name     |   Type    |           Description
*VARNAME        VARTYPE     VARDESCRIPTION
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------
      REAL         A_CVARR  (250)
      CHARACTER*4  A_CVARC  (100)
      INTEGER      A_CVARI  ( 20)

      REAL GetUEFF, LINTRP, DU
      INTEGER MAXIS, CONTROL, IINTERP
      REAL MACH, ALPHA, NPR

      INTEGER IALPHA, IMACH, INPR, IMN, IU, MULT
      REAL C1, C2, C3, C4, C5, C6, C7, C8, D1, D2, D3, D4, D5, D6, D7,
     .     D8, X(2), DC_DU(2), DC_DUF

      INCLUDE 'UEFF_Commons.inc'
! ------------------------------------------------------------------------
!
!     Common Section
!
! ------------------------------------------------------------------------
```

```
      COMMON / A_CVARS / A_CVARR,A_CVARI,A_CVARC

! ---------------------------------------------------------------------
!
!     Equivalence Section
!
! ---------------------------------------------------------------------
      EQUIVALENCE (A_CVARR(3)    , ALPHA           )
      EQUIVALENCE (A_CVARR(61)   , MACH            )
      EQUIVALENCE (A_CVARR(173)  , NPR            )

! ---------------------------------------------------------------------
!
!     Initialization Section
!
! ---------------------------------------------------------------------

! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      IINTERP = 0

      IF (CONTROL .EQ. 1 .OR. CONTROL .EQ. 2) THEN       ! stabilators

      IMACH  = INT((MACH - SMMIN)/SMINC)
      IMACH  = MIN(MAX(0,IMACH),SMI)
      IALPHA = INT((ALPHA - SAMIN)/SAINC)
      IALPHA = MIN(MAX(0,IALPHA),SAI)
      IU     = INT((DU - SDMIN)/SDINC)
      IU     = MIN(MAX(0,IU),SDI)

      IF (MAXIS .EQ. 1) THEN                     ! Cl axis
        IF (CONTROL .EQ. 1) THEN
          MULT = 1
        ELSE
          MULT = -1
        END IF
        C1 = SC1_MAD(IMACH,IU,IALPHA)
        C2 = SC1_MA(IMACH,IU,IALPHA)
        C3 = SC1_MD(IMACH,IU,IALPHA)
        C4 = SC1_M(IMACH,IU,IALPHA)
        C5 = SC1_AD(IMACH,IU,IALPHA)
        C6 = SC1_A(IMACH,IU,IALPHA)
        C7 = SC1_D(IMACH,IU,IALPHA)
        C8 = SC10(IMACH,IU,IALPHA)
      ELSE

        IF (MAXIS .EQ. 2) THEN                     ! Cm axis
          MULT = 1
```

```
    C1 = SCM_MAD(IMACH,IU,IALPHA)
    C2 = SCM_MA(IMACH,IU,IALPHA)
    C3 = SCM_MD(IMACH,IU,IALPHA)
    C4 = SCM_M(IMACH,IU,IALPHA)
    C5 = SCM_AD(IMACH,IU,IALPHA)
    C6 = SCM_A(IMACH,IU,IALPHA)
    C7 = SCM_D(IMACH,IU,IALPHA)
    C8 = SCM0(IMACH,IU,IALPHA)
  ELSE

    IF (MAXIS .EQ. 3) THEN                    ! Cn axis
      IF (CONTROL .EQ. 1) THEN
        MULT = 1
      ELSE
        MULT = -1
      END IF
      C1 = SCN_MAD(IMACH,IU,IALPHA)
      C2 = SCN_MA(IMACH,IU,IALPHA)
      C3 = SCN_MD(IMACH,IU,IALPHA)
      C4 = SCN_M(IMACH,IU,IALPHA)
      C5 = SCN_AD(IMACH,IU,IALPHA)
      C6 = SCN_A(IMACH,IU,IALPHA)
      C7 = SCN_D(IMACH,IU,IALPHA)
      C8 = SCN0(IMACH,IU,IALPHA)
    ELSE

      IF (MAXIS .EQ. 4) THEN                  ! CD axis
        MULT = 1
        C1 = SCD_MAD(IMACH,IU,IALPHA)
        C2 = SCD_MA(IMACH,IU,IALPHA)
        C3 = SCD_MD(IMACH,IU,IALPHA)
        C4 = SCD_M(IMACH,IU,IALPHA)
        C5 = SCD_AD(IMACH,IU,IALPHA)
        C6 = SCD_A(IMACH,IU,IALPHA)
        C7 = SCD_D(IMACH,IU,IALPHA)
        C8 = SCD0(IMACH,IU,IALPHA)
      ELSE

      RETURN

      END IF
    END IF
  END IF
END IF

END IF


IF (CONTROL .EQ. 3 .OR. CONTROL .EQ. 4) THEN       ! ailerons

IMACH  = INT((MACH - AMMIN)/AMINC)
```

```
IMACH  = MIN(MAX(0,IMACH),AMI)
IALPHA = INT((ALPHA - AAMIN)/AAINC)
IALPHA = MIN(MAX(0,IALPHA),AAI)
IU     = INT((DU - ADMIN)/ADINC)
IU     = MIN(MAX(0,IU),ADI)

IF (MAXIS .EQ. 1) THEN                    ! Cl axis
  IF (CONTROL .EQ. 3) THEN
    MULT = 1
  ELSE
    MULT = -1
  END IF
  C1 = AC1_MAD(IMACH,IU,IALPHA)
  C2 = AC1_MA(IMACH,IU,IALPHA)
  C3 = AC1_MD(IMACH,IU,IALPHA)
  C4 = AC1_M(IMACH,IU,IALPHA)
  C5 = AC1_AD(IMACH,IU,IALPHA)
  C6 = AC1_A(IMACH,IU,IALPHA)
  C7 = AC1_D(IMACH,IU,IALPHA)
  C8 = AC10(IMACH,IU,IALPHA)
ELSE

  IF (MAXIS .EQ. 2) THEN                   ! Cm axis
    MULT = 1
    C1 = ACM_MAD(IMACH,IU,IALPHA)
    C2 = ACM_MA(IMACH,IU,IALPHA)
    C3 = ACM_MD(IMACH,IU,IALPHA)
    C4 = ACM_M(IMACH,IU,IALPHA)
    C5 = ACM_AD(IMACH,IU,IALPHA)
    C6 = ACM_A(IMACH,IU,IALPHA)
    C7 = ACM_D(IMACH,IU,IALPHA)
    C8 = ACM0(IMACH,IU,IALPHA)
  ELSE

    IF (MAXIS .EQ. 3) THEN                 ! Cn axis
      IF (CONTROL .EQ. 3) THEN
        MULT = 1
      ELSE
        MULT = -1
      END IF
      C1 = ACN_MAD(IMACH,IU,IALPHA)
      C2 = ACN_MA(IMACH,IU,IALPHA)
      C3 = ACN_MD(IMACH,IU,IALPHA)
      C4 = ACN_M(IMACH,IU,IALPHA)
      C5 = ACN_AD(IMACH,IU,IALPHA)
      C6 = ACN_A(IMACH,IU,IALPHA)
      C7 = ACN_D(IMACH,IU,IALPHA)
      C8 = ACN0(IMACH,IU,IALPHA)
    ELSE

      IF (MAXIS .EQ. 4) THEN               ! CD axis
```

```
        MULT = 1
        C1 = ACD_MAD(IMACH,IU,IALPHA)
        C2 = ACD_MA(IMACH,IU,IALPHA)
        C3 = ACD_MD(IMACH,IU,IALPHA)
        C4 = ACD_M(IMACH,IU,IALPHA)
        C5 = ACD_AD(IMACH,IU,IALPHA)
        C6 = ACD_A(IMACH,IU,IALPHA)
        C7 = ACD_D(IMACH,IU,IALPHA)
        C8 = ACD0(IMACH,IU,IALPHA)
      ELSE

      RETURN

      END IF
    END IF
  END IF
END IF

END IF

IF (CONTROL .EQ. 5) THEN                          ! rudder

IMACH  = INT((MACH - RMMIN)/RMINC)
IMACH  = MIN(MAX(0,IMACH),RMI)
IALPHA = INT((ALPHA - RAMIN)/RAINC)
IALPHA = MIN(MAX(0,IALPHA),RAI)
IU     = INT((DU - RDMIN)/RDINC)
IU     = MIN(MAX(0,IU),RDI)

IF (MAXIS .EQ. 1) THEN                    ! Cl axis
  MULT = 1
  C1 = RC1_MAD(IMACH,IU,IALPHA)
  C2 = RC1_MA(IMACH,IU,IALPHA)
  C3 = RC1_MD(IMACH,IU,IALPHA)
  C4 = RC1_M(IMACH,IU,IALPHA)
  C5 = RC1_AD(IMACH,IU,IALPHA)
  C6 = RC1_A(IMACH,IU,IALPHA)
  C7 = RC1_D(IMACH,IU,IALPHA)
  C8 = RC10(IMACH,IU,IALPHA)
ELSE

  IF (MAXIS .EQ. 2) THEN                    ! Cm axis
    MULT = 1
    C1 = RCM_MAD(IMACH,IU,IALPHA)
    C2 = RCM_MA(IMACH,IU,IALPHA)
    C3 = RCM_MD(IMACH,IU,IALPHA)
    C4 = RCM_M(IMACH,IU,IALPHA)
    C5 = RCM_AD(IMACH,IU,IALPHA)
    C6 = RCM_A(IMACH,IU,IALPHA)
    C7 = RCM_D(IMACH,IU,IALPHA)
    C8 = RCM0(IMACH,IU,IALPHA)
```

```
  ELSE

    IF (MAXIS .EQ. 3) THEN                    ! Cn axis
      MULT = 1
      C1 = RCN_MAD(IMACH,IU,IALPHA)
      C2 = RCN_MA(IMACH,IU,IALPHA)
      C3 = RCN_MD(IMACH,IU,IALPHA)
      C4 = RCN_M(IMACH,IU,IALPHA)
      C5 = RCN_AD(IMACH,IU,IALPHA)
      C6 = RCN_A(IMACH,IU,IALPHA)
      C7 = RCN_D(IMACH,IU,IALPHA)
      C8 = RCN0(IMACH,IU,IALPHA)
    ELSE

      IF (MAXIS .EQ. 4) THEN                  ! CD axis
        MULT = 1
        C1 = RCD_MAD(IMACH,IU,IALPHA)
        C2 = RCD_MA(IMACH,IU,IALPHA)
        C3 = RCD_MD(IMACH,IU,IALPHA)
        C4 = RCD_M(IMACH,IU,IALPHA)
        C5 = RCD_AD(IMACH,IU,IALPHA)
        C6 = RCD_A(IMACH,IU,IALPHA)
        C7 = RCD_D(IMACH,IU,IALPHA)
        C8 = RCD0(IMACH,IU,IALPHA)
      ELSE

      RETURN

      END IF
    END IF
  END IF
END IF

END IF

IF (CONTROL .EQ. 6 .OR. CONTROL .EQ. 7) THEN       ! canards

IMACH  = INT((MACH - CMMIN)/CMINC)
IMACH  = MIN(MAX(0,IMACH),CMI)
IALPHA = INT((ALPHA - CAMIN)/CAINC)
IALPHA = MIN(MAX(0,IALPHA),AAI)
IU     = INT((DU - CDMIN)/CDINC)
IU     = MIN(MAX(0,IU),CDI)

IF (MAXIS .EQ. 1) THEN                    ! Cl axis
  IF (CONTROL .EQ. 6) THEN
    MULT = 1
  ELSE
    MULT = -1
  END IF
  C1 = CC1_MAD(IMACH,IU,IALPHA)
```

```
  C2 = CC1_MA(IMACH,IU,IALPHA)
  C3 = CC1_MD(IMACH,IU,IALPHA)
  C4 = CC1_M(IMACH,IU,IALPHA)
  C5 = CC1_AD(IMACH,IU,IALPHA)
  C6 = CC1_A(IMACH,IU,IALPHA)
  C7 = CC1_D(IMACH,IU,IALPHA)
  C8 = CC10(IMACH,IU,IALPHA)
ELSE

  IF (MAXIS .EQ. 2) THEN                    ! Cm axis
    MULT = 1
    C1 = CCM_MAD(IMACH,IU,IALPHA)
    C2 = CCM_MA(IMACH,IU,IALPHA)
    C3 = CCM_MD(IMACH,IU,IALPHA)
    C4 = CCM_M(IMACH,IU,IALPHA)
    C5 = CCM_AD(IMACH,IU,IALPHA)
    C6 = CCM_A(IMACH,IU,IALPHA)
    C7 = CCM_D(IMACH,IU,IALPHA)
    C8 = CCM0(IMACH,IU,IALPHA)
  ELSE

    IF (MAXIS .EQ. 3) THEN                  ! Cn axis
      IF (CONTROL .EQ. 6) THEN
        MULT = 1
      ELSE
        MULT = -1
      END IF
      C1 = CCN_MAD(IMACH,IU,IALPHA)
      C2 = CCN_MA(IMACH,IU,IALPHA)
      C3 = CCN_MD(IMACH,IU,IALPHA)
      C4 = CCN_M(IMACH,IU,IALPHA)
      C5 = CCN_AD(IMACH,IU,IALPHA)
      C6 = CCN_A(IMACH,IU,IALPHA)
      C7 = CCN_D(IMACH,IU,IALPHA)
      C8 = CCN0(IMACH,IU,IALPHA)
    ELSE

      IF (MAXIS .EQ. 4) THEN                ! CD axis
        MULT = 1
        C1 = CCD_MAD(IMACH,IU,IALPHA)
        C2 = CCD_MA(IMACH,IU,IALPHA)
        C3 = CCD_MD(IMACH,IU,IALPHA)
        C4 = CCD_M(IMACH,IU,IALPHA)
        C5 = CCD_AD(IMACH,IU,IALPHA)
        C6 = CCD_A(IMACH,IU,IALPHA)
        C7 = CCD_D(IMACH,IU,IALPHA)
        C8 = CCD0(IMACH,IU,IALPHA)
      ELSE

      RETURN
```

```
      END IF
    END IF
  END IF
END IF

END IF

IF (CONTROL .EQ. 8) THEN                        ! Yaw Nozzle

IMACH  = INT((MACH - YNMMIN)/YNMINC)
IMACH  = MIN(MAX(0,IMACH),YNMI)
IALPHA = INT((ALPHA - YNAMIN)/YNAINC)
IALPHA = MIN(MAX(0,IALPHA),YNAI)
IU     = INT((DU - YNDMIN)/YNDINC)
IU     = MIN(MAX(0,IU),YNDI)
INPR   = INT((NPR - 1.0)/4.0)
INPR   = MIN(MAX(0,INPR),6)

IF (MAXIS .EQ. 1) THEN                     ! Cl axis
  GetUEFF = 0.
  RETURN
ELSE

  IF (MAXIS .EQ. 2) THEN                     ! Cm axis
    GetUEFF = 0.
    RETURN
  ELSE

    IF (MAXIS .EQ. 3) THEN                     ! Cn axis
      IINTERP = 1          ! Interpolate if we can
      MULT = 1
      IMN    = INPR*(YNMI + 1) + IMACH
      C1 = YNCN_MAD(IMN,IU,IALPHA)
      C2 = YNCN_MA(IMN,IU,IALPHA)
      C3 = YNCN_MD(IMN,IU,IALPHA)
      C4 = YNCN_M(IMN,IU,IALPHA)
      C5 = YNCN_AD(IMN,IU,IALPHA)
      C6 = YNCN_A(IMN,IU,IALPHA)
      C7 = YNCN_D(IMN,IU,IALPHA)
      C8 = YNCN0(IMN,IU,IALPHA)
      IF (INPR .LT. 6 .AND. IINTERP .EQ. 1) THEN
        INPR = INPR + 1
        IMN    = INPR*(YNMI + 1) + IMACH
        D1 = YNCN_MAD(IMN,IU,IALPHA)
        D2 = YNCN_MA(IMN,IU,IALPHA)
        D3 = YNCN_MD(IMN,IU,IALPHA)
        D4 = YNCN_M(IMN,IU,IALPHA)
        D5 = YNCN_AD(IMN,IU,IALPHA)
        D6 = YNCN_A(IMN,IU,IALPHA)
        D7 = YNCN_D(IMN,IU,IALPHA)
        D8 = YNCN0(IMN,IU,IALPHA)
```

```
      ELSE
        IINTERP = 0
      END IF
    ELSE

      IF (MAXIS .EQ. 4) THEN                  ! CD axis
        GetUEFF = 0.
        RETURN
      ELSE

      RETURN

      END IF
    END IF
  END IF
END IF

END IF

IF (CONTROL .EQ. 9) THEN                      ! Pitch Nozzle

IMACH  = INT((MACH - PNMMIN)/PNMINC)
IMACH  = MIN(MAX(0,IMACH),PNMI)
IALPHA = INT((ALPHA - PNAMIN)/PNAINC)
IALPHA = MIN(MAX(0,IALPHA),PNAI)
IU     = INT((DU - PNDMIN)/PNDINC)
IU     = MIN(MAX(0,IU),PNDI)
INPR   = INT((NPR - 1.0)/4.0)
INPR   = MIN(MAX(0,INPR),6)

IF (MAXIS .EQ. 1) THEN                    ! Cl axis
  GetUEFF = 0.
  RETURN
ELSE

  IF (MAXIS .EQ. 2) THEN                    ! Cm axis
    IINTERP = 1          ! Interpolate if we can
    MULT = 1
    IMN    = INPR*(PNMI + 1) + IMACH
    C1 = PNCM_MAD(IMN,IU,IALPHA)
    C2 = PNCM_MA(IMN,IU,IALPHA)
    C3 = PNCM_MD(IMN,IU,IALPHA)
    C4 = PNCM_M(IMN,IU,IALPHA)
    C5 = PNCM_AD(IMN,IU,IALPHA)
    C6 = PNCM_A(IMN,IU,IALPHA)
    C7 = PNCM_D(IMN,IU,IALPHA)
    C8 = PNCM0(IMN,IU,IALPHA)
    IF (INPR .LT. 6 .AND. IINTERP .EQ. 1) THEN
      INPR = INPR + 1
      IMN    = INPR*(PNMI + 1) + IMACH
      D1 = PNCM_MAD(IMN,IU,IALPHA)
```

```
      D2 = PNCM_MA(IMN,IU,IALPHA)
      D3 = PNCM_MD(IMN,IU,IALPHA)
      D4 = PNCM_M(IMN,IU,IALPHA)
      D5 = PNCM_AD(IMN,IU,IALPHA)
      D6 = PNCM_A(IMN,IU,IALPHA)
      D7 = PNCM_D(IMN,IU,IALPHA)
      D8 = PNCM0(IMN,IU,IALPHA)
    ELSE
      IINTERP = 0
    END IF
  ELSE

    IF (MAXIS .EQ. 3) THEN                    ! Cn axis
      GetUEFF = 0.
      RETURN
    ELSE

      IF (MAXIS .EQ. 4) THEN                  ! CD axis
        IINTERP = 1          ! Interpolate if we can
        MULT = 1
        IMN    = INPR*(PNMI + 1) + IMACH
        C1 = PNCD_MAD(IMN,IU,IALPHA)
        C2 = PNCD_MA(IMN,IU,IALPHA)
        C3 = PNCD_MD(IMN,IU,IALPHA)
        C4 = PNCD_M(IMN,IU,IALPHA)
        C5 = PNCD_AD(IMN,IU,IALPHA)
        C6 = PNCD_A(IMN,IU,IALPHA)
        C7 = PNCD_D(IMN,IU,IALPHA)
        C8 = PNCD0(IMN,IU,IALPHA)
        IF (INPR .LT. 6 .AND. IINTERP .EQ. 1) THEN
          INPR = INPR + 1
          IMN    = INPR*(PNMI + 1) + IMACH
          D1 = PNCD_MAD(IMN,IU,IALPHA)
          D2 = PNCD_MA(IMN,IU,IALPHA)
          D3 = PNCD_MD(IMN,IU,IALPHA)
          D4 = PNCD_M(IMN,IU,IALPHA)
          D5 = PNCD_AD(IMN,IU,IALPHA)
          D6 = PNCD_A(IMN,IU,IALPHA)
          D7 = PNCD_D(IMN,IU,IALPHA)
          D8 = PNCD0(IMN,IU,IALPHA)
        ELSE
          IINTERP = 0
        END IF
      ELSE

      RETURN

      END IF
    END IF
  END IF
END IF
```

```
      END IF

      DC_DU(1) = MACH*ALPHA*(C1*DU + C2) + MACH*(C3*DU + C4) +
     .               ALPHA*(C5*DU + C6) + C7*DU + C8

      IF (IINTERP .EQ. 1) THEN
        DC_DU(2) = MACH*ALPHA*(D1*DU + D2) + MACH*(D3*DU + D4) +
     .                  ALPHA*(D5*DU + D6) + D7*DU + D8
        X(1) = (INPR - 1)*4.0 + 1
        X(2) = INPR*4.0 + 1
        DC_DUF = LINTRP(X,DC_DU,NPR)
      ELSE
        DC_DUF = DC_DU(1)
      END IF


      GetUEFF = MULT*DC_DUF

! ----------------------------------------------------------------------
!
!     End Of: GetUEFF
!
! ----------------------------------------------------------------------
      RETURN
      END

C INTERPOLATION FUNCTION
      FUNCTION LINTRP(X,F,XD)
      REAL LINTRP,X(2),F(2),XD
      LINTRP = (XD - X(2))/(X(1) - X(2))*(F(1) - F(2)) + F(2)
      RETURN
      END
```

2.2 Control Constraints

A. Usage

This function is used to calculate vectors of maximum and minimum position constraints and rate limit constraints for positive and negative deflection directions.

A.1 GETCSTR

Function Prototype, SUBROUTINE

INTEGER	M, IU
REAL	UMAX, UMIN, URMAX, URMIN

Assign data to M and IU.

CALL GETCSTR(M, IU, UMAX, UMIN, URMAX, URMIN)

GETCSTR returns the vectors of maximum and minimum position constraints in UMAX and UMIN, and returns rate limits in the positive and negative deflection directions in URMAX and URMIN.

Argument Definitions

M	[in]	The number of controls in the allocatable controls vector.
IU	[in]	An m-Dimensional vector of control indices that relates the allocatable controls vector to the vector of physical aircraft controls.
UMAX	[out]	Vector of maximum deflection position limits.
UMIN	[out]	Vector of minimum deflection position limits.
URMAX	[out]	Vector of rate limits in the positive deflection direction.
URMIN	[out]	Vector of rate limits in the negative deflection direction.

B. General Remarks

Because of the reconfigurable nature of CARL, the IU vector is needed to map the allocatable controls vector to the aircraft controls vector. As an example, if the left and right ailerons are controls 3 and 4 in both the aircraft controls vector and allocatable controls vector then IU(3) and IU(4) will be 3 and 4 respectively. If at some time, the left aileron is dropped from the allocation procedure because of a failure, then the 3rd allocatable control will be the right aileron, and IU(3) will then be 4.

C. Functional Description

This subroutine begins by setting the absolute minimum and maximum constraints for the 7 aerodynamic controls. For the yaw and pitch thrust vectoring nozzles (aircraft controls 8 and 9), the maximum and minimum deflection limits are found as a function of engine thrust according to Section 8.3. Both yaw and pitch nozzles are given equal priority. Following the calculations of the position limits, the rate limits for the positive and negative deflection directions are assigned for all 9 controls.

D. Errors and Restrictions

With the exception of the thrust vectoring nozzles, this subroutine assumes that the position and rate limits are constant and equal to the nominal no-load values. It is also assumed that both engines are producing the same amount of thrust. Therefore, if one engine is shut down, then there is a chance that the position constraints

calculated will violate the 4000 lb. radial force limit. The maximum number of controls allowed is 20.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: GetCSTR
!       Called By: CONALLO (or A_C$GETCSTR)
!       Calls to: none
!
! ------------------------------------------------------------------------
      SUBROUTINE GetCSTR(M, IU, UMAX, UMIN, URMAX, URMIN)
! ------------------------------------------------------------------------
!
!       Function:   Sets up the control constraints for the Control
!                Allocation stuff.
!
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                           By
!  MAR 23 1997        deleted the URATE argument and replaced it with
!                     URMIN and URMAX to account for the fact that the
!                     controls may be able to move faster in one direc-
!                     tion than the other.                          JB
!  MAR  08 1997       Revised the argument list and removed the section
!                     where the actual constraints that conallo sees
!                     are calculated. This subroutine now just returns
!                     position limits and rate limits and lets conallo
!                     figure out what the actual constraints are.   J.B.
!  SEPT 11 1996       Created so that the contraints in the Control
!                     Allocation subroutines could become aircraft
!                     (and/or aircraft state) dependant             J.B.
!
! ------------------------------------------------------------------------
!
!     Glossary Section
!
! ------------------------------------------------------------------------
!                            Global Variables
!
!     Name     |   Type    |           Description
*URATE0(1)      REAL        Nominal def. rate (deg/s)
*UMIN0(1)       REAL        Def. Limit (deg)
*UMAX0(1)       REAL        Def. limit (deg)
*ENGTHRUST      REAL        Total Engine thrust
!
!                            Local Variables
!
```

```
!     Name     |   Type    |           Description
*VARNAME        VARTYPE     VARDESCRIPTION
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------
      INTEGER Max_Controls
      PARAMETER (Max_Controls = 20)

      REAL         A_CVARR  (250)
      CHARACTER*4  A_CVARC  (100)
      INTEGER      A_CVARI  ( 20)

      REAL UMIN0(Max_Controls),UMAX0(Max_Controls),
     .     URATE0(Max_Controls), ENGTHRUST, RTOD
! -------------------------------Locals-----------------------------------
      LOGICAL Use_Globals
      INTEGER IU(Max_Controls),M,I
      REAL UMAX(Max_Controls),UMIN(Max_Controls),URMAX(Max_Controls),
     .     URMIN(Max_Controls)
      REAL THRUST,RA_T_LIMIT,K,RMAX
      PARAMETER (RA_T_LIMIT = 4000)       ! radial force limit
! ------------------------------------------------------------------------
!
!     Common Section
!
! ------------------------------------------------------------------------
      COMMON / A_CVARS / A_CVARR,A_CVARI,A_CVARC

! ------------------------------------------------------------------------
!
!     Equivalence Section
!
! ------------------------------------------------------------------------
      EQUIVALENCE (A_CVARR(84)   , URATE0(1)      )
      EQUIVALENCE (A_CVARR(124)  , UMIN0(1)       )
      EQUIVALENCE (A_CVARR(104)  , UMAX0(1)       )
      EQUIVALENCE (A_CVARR(174)  , ENGTHRUST      )

! ------------------------------------------------------------------------
!
!     Initialization Section
!
! ------------------------------------------------------------------------

! ------------------------------------------------------------------------
!
!     Run Section
```

```
!
! ------------------------------------------------------------------
      THRUST = 0.5*ENGTHRUST
      RTOD = 57.2958

      DO 1010 I = 1,M

! Position Limits for aero controls

        IF (IU(I) .NE. 8 .AND. IU(I) .NE. 9) THEN   ! aero controls.
          UMIN(I) = UMIN0(IU(I))
          UMAX(I) = UMAX0(IU(I))
        ELSE

! Make sure that we don't violate the 4000 lb radial thrust limit. First
! find the maximum "radial" deflection allowed. Then find PNOZmax and
! YNOZmax such that the constraint box lies entirely within the bounding
! circle.

            IF (THRUST .GE. RA_T_LIMIT) THEN
              RMAX = ASIN(RA_T_LIMIT/THRUST)*RTOD
              RMAX = AMIN1(RMAX,20.0)
            ELSE
              RMAX = 20.0
            END IF

! scale RMAX
            RMAX = RMAX*SQRT(2.0)/2.0

! Thrust vectoring limits

            UMIN(I) = -RMAX
            UMAX(I) =  RMAX

        END IF

! Rate Limits

        URMAX(I) = URATE0(IU(I))
        URMIN(I) = URATE0(IU(I))

1010  CONTINUE
! ------------------------------------------------------------------
!
!     End of GetCSTR
!
! ------------------------------------------------------------------
      RETURN
      END
```

APPENDIX IV.

Shell Interface Routines

This Appendix contains documentation on the Shell Interface routines required for the SweepData utility described in Appendix I.

1. Text Justification Utilities

A. Usage

These functions can be used to perform various text justification styles to character strings.

A.1 GETLEFTJUSTIFY

Function Prototype, CHARACTER*(*)
CHARACTER*(*) TEXTFIELD

Assign a CHARACTER string to TEXTFIELD

TEXTLEFT = GETLEFTJUSTIFY(TEXTFIELD)

The CHARACTER string returned has all of the leading spaces deleted and the string shifted accordingly

Argument Definitions

TEXTFIELD	[in]	Any CHARACTER variable containing ASCII data

A.2 GETCENTERJUSTIFY

Function Prototype, CHARACTER*(*)
CHARACTER*(*) TEXTFIELD
INTEGER*4 SCRN_W, TEXT_W, ISPACE

Assign values to all variables

TEXTCENTER = GETCENTERJUSTIFY(TEXTFIELD, SCRN_W, TEXT_W, ISPACE)

The CHARACTER string returned has various forms of center justification based on the value of ISPACE

Argument Definitions

TEXTFIELD	[in]	Any CHARACTER variable containing ASCII data
SCRN_W	[in]	The width (in # of characters) of the screen that the text is to be centered in.
TEXT_W	[in]	The length (in bytes) of the character variable TEXTFIELD.
ISPACE	[in]	a value of 0 implies to treat trailing spaces as being significant so that the entire TEXTFIELD is centered. A value of 1 implies to drop the trailing spaces and center just the bytes containing ASCII characters.

B. General Remarks

Many compilers now have extensions for removing leading spaces from character data. In fact, the method used in all of the low level routines for the Shell use the Language Systems extension ADJUSTL. The GETLEFTJUSTIFY function is still included however to allow other compilers that may not have this extension to use this feature. If this is the case, then the programmer may alter the name of this function to ADJUSTL instead of having to change every call statement throughout the code.

C. Functional Description

The GETLEFTJUSTIFY function scans each byte in the data string to determine if it is a space or not. If it is, then the function continues to the next byte and checks again. If the current byte does not contain a space, then the function returns, as a result, a data string whose first character is held by that byte.
The GETCENTERJUSTIFY function operates in a similar fashion when trailing spaces are treated as being significant (ISPACE = 0). It calculates the starting byte of the result by dividing the difference between the screen width and text width by 2. When the trailing spaces are not significant, the function first finds the byte at which 2 or more spaces occur. This index is then subtracted from the screen width and divided by 2 to get the starting byte for the result. Because the beginning bytes of the result are not specified in this function, they are padded with spaces so that when the result is printed to the screen, it appears to be centered.

D. Errors and Restrictions

When using the GETCENTERJUSTIFY function, it is important that the text data have no multiple spaces in sequence. Otherwise, the data will be truncated. The ISPACE integer must also be 0 or 1. If it is not, then the GETCENTERJUSTIFY function will return a character string full of spaces. The TEXT_W field should always be less than or equal to the SCRN_W field or an error will occur. The programmer should also make sure that both the variable expecting the result and the function itself are declared with at least the same length as the screen width field. Otherwise, the centered data may be truncated, or it may not appear at all, resulting in an error.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: GetLEFTJustify
!           Called By: various SHell and FSGI routines
!           Calls to: none
!
! ------------------------------------------------------------------------
      FUNCTION GetLeftJustify(Text_Field)
! ------------------------------------------------------------------------
!
!       Function:   Given a text field, this function returns a text
!                string whose leading spaces have been truncated.
!
!       *NOTE*   This function is only supported for backwards
!                compatibility. Shell Versions 1.4 and above should use
!                the extension ADJUSTL.
!
! ------------------------------------------------------------------------
!
```

```
!       Modifications:
!     Date                        Purpose                          By
!  FEB 09 1996        Created                                      J.B.
!  MAR 12 1996        Modified for Shell V1.5. *NOTE* not required
!                     for V1.4 and above.                          J.B.
!
! ----------------------------------------------------------------------
      IMPLICIT NONE
! ----------------------------------------------------------------------
!
!     Declaration Section
!
! ----------------------------------------------------------------------
      CHARACTER*(*) GetLEFTJustify
      CHARACTER*(*) Text_Field
      INTEGER Cindx
      LOGICAL Cindxfound
! ----------------------------------------------------------------------
!
!     Run Section
!
! ----------------------------------------------------------------------
      Cindxfound = .FALSE.
      Cindx = 1

      DO WHILE (.not. Cindxfound)
        IF (Text_Field(Cindx:Cindx) .NE. ' ') Cindxfound = .TRUE.
        Cindx = Cindx + 1
      END DO

      GetLEFTJustify = Text_Field(Cindx:)
! ----------------------------------------------------------------------
!
!     End of GetLEFTJustify
!
! ----------------------------------------------------------------------
      RETURN
      END
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ----------------------------------------------------------------------
!
!       Module Name: GetCENTERJustify
!           Called By: various SHell and FSGI routines
!           Calls to: none
!
! ----------------------------------------------------------------------
      FUNCTION GetCENTERJustify(Text_field,Scr_W,Text_W,ISpace)
! ----------------------------------------------------------------------
!
!       Function:   Given a text field, a screen width, and the width
!                 of the text field, this function returns a text string
```

```
!               padded with the appropriate number of leading spaces
!               such that the string is centered in the middle of the
!               screen.
!
!      ISpace is an integer which determines how trailing spaces are
!             handled:
!             0 = treat spaces as if they are actual characters
!             1 = remove trailing spaces and center the left over string
!                 (This is the "True" Center justification of a string)
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                          By
!  APR 09 1996        Created                                     J.B.
!
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------
      CHARACTER*(*) GetCENTERJustify
      INTEGER Scr_W,Text_W,ISpace
      CHARACTER*(*) Text_Field
      INTEGER Charindex
! ------------------------------------------------------------------------
!
!     Run Section
!
! ------------------------------------------------------------------------
      GetCENTERJustify = ' '
      SELECT CASE (ISpace)

        CASE (0)           ! don't truncate spaces

          Charindex = INT((Scr_W - Text_W)/2.0)
          GetCENTERJustify(Charindex:) = Text_Field

        CASE (1)           ! truncate spaces first

          Charindex = INDEX(Text_Field,'  ') - 1
          Charindex = INT((Scr_W - Charindex)/2.0)
          GetCENTERJustify(Charindex:) = Text_Field

      END SELECT
! ------------------------------------------------------------------------
!
!     End of GetCENTERJustify
!
```

```
! ---------------------------------------------------------------------
      RETURN
      END
```

2. Text Conversions

A. Usage

This category of functions allows conversions of different data types into character strings.

A.1 LTOSTRING

Function Prototype, CHARACTER*1
LOGICAL LVAR

Assign a value to LVAR

LSTRING = LTOSTRING(LVAR)

The CHARACTER string returned is “T” if LVAR is TRUE and “F” if LVAR is FALSE

Argument Definitions

LVAR	[in]	Any LOGICAL variable

A.2 I4TOSTRING

Function Prototype, CHARACTER*8
INTEGER*4 IVAR

Assign a value to IVAR

ISTRING = I4TOSTRING(IVAR)

The CHARACTER string returned contains the INTEGER value represented as ASCII text data (left justified).

Argument Definitions

IVAR	[in]	Any INTEGER variable

A.3 R4TOSTRING

Function Prototype, CHARACTER*16
REAL*4 RVAR
CHARACTER*3 FORMAT

Assign values to all variables

RSTRING = R4TOSTRING(RVAR, FORMAT)

The character string returned contains the Real value represented as ASCII text in either a fixed point notation or an exponential notation (left justified)

Argument Definitions

RVAR	[in]	Any REAL variable
FORMAT	[in]	Represents the format to be converted to.

B. General Remarks

All of these functions use an internal WRITE statement to save the data to a buffer. They then read this buffer and

left justify the data using ADJUSTL. They are often used to place numeric data into a string of longer text before it is sent to other string processing procedures. When using the R4TOSTRING function it is acceptable to replace character literals for the FORMAT argument in the call statement. Using 'fix' produces a result in fixed point notation, while 'sci' will result in the data being represented in scientific notation.

C. Functional Description

The LTOSTRING and I4TOSTRING functions simply write the input data to an internal CHARACTER buffer. Then they left justify the data using the ADJUSTL (or optionally GETLEFTJUSTIFY) function and return the result. The R4TOSTRING function first checks the FORMAT argument. If FORMAT is 'fix' or 'FIX' it returns the data in a fixed point notation. Otherwise, the data is returned in scientific notation.

D. Errors and Restrictions

When using these functions make sure that the data passed to them represents the data that they expect. Otherwise, the results returned (if an error does not occur) will be meaningless. The R4TOSTRING function currently supports only fixed and scientific notations formatted as F16.5 and E16.5 respectively. Future versions may support more options, but as of this writing, these formats have been sufficient.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: LtoString
!           Called By: Various string processing requests
!           Calls to: none
!
! ------------------------------------------------------------------------
      FUNCTION LtoString (DALOGICAL)
! ------------------------------------------------------------------------
!
!       Function:   Converts a logical variable to a character*1 value
!
!
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------
      CHARACTER*1 LtoString
      LOGICAL DALOGICAL
      CHARACTER*1 Buffer
! ------------------------------------------------------------------------
!
!     Run Section
!
! ------------------------------------------------------------------------
```

```
      WRITE (Buffer,'(L1)') DALOGICAL
      LtoString = Buffer
! ---------------------------------------------------------------------
!
!     End of LtoString
!
! ---------------------------------------------------------------------
      RETURN
      END
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ---------------------------------------------------------------------
!
!       Module Name: I4toString
!           Called By: Various string processing requests
!           Calls to: none
!
! ---------------------------------------------------------------------
      FUNCTION I4toString (DAVALUE)
! ---------------------------------------------------------------------
!
!       Function:   Converts a INTEGER*4 variable to a character*8
!                value
!
!
! ---------------------------------------------------------------------
      IMPLICIT NONE
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
      CHARACTER*8 I4toString
      INTEGER*4 DAVALUE
      CHARACTER*8 Buffer
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      WRITE (Buffer,'(I8)') DAVALUE
      I4toString = ADJUSTL(Buffer)
! ---------------------------------------------------------------------
!
!     End of I4toString
!
! ---------------------------------------------------------------------
      RETURN
      END
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ---------------------------------------------------------------------
!
```

```
!       Module Name: R4toString
!           Called By: Various string processing requests
!           Calls to: none
!
! ---------------------------------------------------------------------
      FUNCTION R4toString (DAVALUE,Format)
! ---------------------------------------------------------------------
!
!       Function:   Converts a REAL*4 variable to a character*16 value
!                Format specifies how the value is written and can be:
!                  'fix'   (Fixed decimal point; 5 decimal places)
!                  'sci'   (Scientific notation)
!
!
! ---------------------------------------------------------------------
      IMPLICIT NONE
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
      CHARACTER*16 R4toString
      REAL*4 DAVALUE
      CHARACTER*3 Format
      CHARACTER*16 Buffer
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      IF (Format .EQ. 'FIX' .OR. Format .EQ. 'fix') THEN
        WRITE (Buffer, '(F16.5)') DAVALUE
      ELSE
        WRITE (Buffer, '(E16.5)') DAVALUE
      END IF
      R4toString = ADJUSTL(Buffer)
! ---------------------------------------------------------------------
!
!     End of R4toString
!
! ---------------------------------------------------------------------
      RETURN
      END
```

3. String Processing Functions

A. Usage

String processing functions allow ways of getting general messages to and from the Shell.

A.1 PROCESSIOSTRING

Function Prototype, INTEGER*4

CHARACTER*80	MESSAGE(NL)
INTEGER*4	NL, MSTAT, MTYPE

Assign values to all arguments

```
IOSTAT = PROCESSIOSTRING(MESSAGE, NL, MSTAT, MTYPE)
```

The character strings in MESSAGE are displayed and the function waits for user interaction depending on the values of MSTAT. I/O errors, if any, are returned in IOSTAT.

Argument Definitions

MESSAGE	[in]	A CHARACTER array containing the text to display.
NL	[in]	The number of text lines in MESSAGE.
MSTAT	[in]	an integer representing the “status code” of the message (see Remarks).
MTYPE	[in]	an integer representing the “type code” of the message (see Remarks).

B. General Remarks

The PROCESSIOSTRING function is the primary means of getting miscellaneous information to and from the user. It is responsible for determining when the Shell is active and when the graphical frontend is active and displays the text accordingly. When the Shell is active, it writes the text to the screen, and when the Frontend is active, it displays the text in an appropriate dialog box. The MSTAT and MTYPE parameters determine the nature of the messages. These are defined below:

- MSTAT = 0 Send the messages to the screen and return
- MSTAT = 1 Wait for a message from the user. The input string is returned in MESSAGE(1)
- MSTAT = 2 Send one message to the screen, wait for a reply. The reply is returned in MESSAGE(2)
- MSTAT = 3 Send one message to the screen and suppress the carriage return. (The next message sent to the screen will appear on the same line.)
- MSTAT = 4 Send Messages to the screen, wait for a y/n reply. (The reply is returned as the last message in the array.)

- MTYPE = 0 Regular message (no priority) Used in Shell mode only.
- MTYPE = 1 Message has notification priority. Implies that no serious errors will occur by continuing and ignoring the message.
- MTYPE = 2 Message has Warning/Caution priority. Implies that an error may or may not occur if the message is ignored.
- MTYPE = 3 Message has STOP priority. Display this message before the Fortran statement STOP has occurred.
- MTYPE = 4 Error Message. Use this priority to describe an error.

By using various combinations of MSTAT and MTYPE, or using multiple calls to PROCESSIOSTRING, practically any effect can be achieved. It is a very powerful function and should be used whenever possible.

C. Functional Description

This function first adds the appropriate header to the beginning of the first line of the message depending on the value of MTYPE. These are '(no header)', 'NOTE:', 'WARNING:', 'STOP:', and 'ERROR:' for MTYPE of 0, 1, 2, 3, and 4 respectively. It then utilizes the SELECT CASE/END SELECT structure to determine what action to take in response to the status code, and whether or not the Shell is active or the Frontend is active.

D. Errors and Restrictions

When calling this function, the MSTAT and MTYPE arguments must be specified correctly. If they are not, then the function simply returns without sending any output to the screen. Be warned that when the Frontend is active, this function calls two additional functions, FE_SHELLALERT and FE_SHELLREPLY which use the Macintosh toolbox. If this function is to be used on another platform without the frontend, then this section could be deleted. (Although it is recommended that these routines just be replaced by dummy routines). Be warned that when calling PROCESSIOSTRING with an MSTAT of 2, The MESSAGE argument must be an array with atleast 2 entries since the user response is returned as MESSAGE(2).

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: ProcessIOString
!           Called By: Shell and FSGI modules
!           Calls to: none
!
! ------------------------------------------------------------------------
      FUNCTION ProcessIOString(Message,Lines,MStatus,MType)
! ------------------------------------------------------------------------
!
!       Function:   Reads a text string and either sends it to the
!                   Shell window or to a dialog box, or reads in a
!                   generic input string.
!
!       CallerID: 0 = Shell mode, 1 = FSGI mode
!       MStatus:  0 = Output message
!                 1 = Receive a message
!                 2 = Output message, expect reply
!                 3 = Output Message, suppress <CR>
!                 4 = Output Message, expect y/n reply.
!       MType:    0 = NIL (used in Shell mode only)
!                 1 = Note alert box
!                 2 = Warning/Caution alert box
!                 3 = Stop alert box
!                 4 = Error message
!       Lines:    number of lines in message
!
!
```

```
! ---------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                          By
!  MAR 28 1996        Created                                John Bolling
!  APR 09 1996        Added capability for Mstatus of 2 and Mtype
!                 of 4.                                              J.B.
!  MAY 16 1996        Added Did_Menu LOGICAL to SHELLPARMS Common
!                 for better communication between the Shell interface
!                 and the GetMenuText module.                        J.B.
!  JAN 20 1997        Changed the Message() data type to CHARACTER*80
!                 to allow an easier port to other platforms.        J.B.
!
! ---------------------------------------------------------------------
      IMPLICIT NONE
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
! ----------------------------Shell Parameters-------------------------
      INTEGER CallerID,Mode
      LOGICAL Do_Menu,Did_Menu
      CHARACTER*80 CommandBuffer
! -----------------------------Locals----------------------------------
      LOGICAL EndString
      INTEGER I,MStatus,ProcessIOString,MType,Lines, CINDX, IOCODE
      CHARACTER*80 Message(Lines), OutWinString
      CHARACTER*9 MHeader(0:4)
      CHARACTER*8 Prompt
      PARAMETER (Prompt = ' ')
! ----------------------------frontend specific------------------------
      INTEGER FE_SHELLAlert
      CHARACTER*1 FE_SHELLReply, Reply
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / SHELLPARMS / CallerID, Mode, Do_Menu, Did_Menu,
     .                      CommandBuffer

! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      MHeader(0) = '         '
      MHeader(1) = 'NOTE:    '
      MHeader(2) = 'WARNING: '
      MHeader(3) = 'STOP:    '
```

```
      MHeader(4) = 'ERROR:   '

      SELECT CASE (CallerID)

        CASE (1)                                  ! FSGI mode
!
!         Under Construction...
!
          SELECT CASE (MStatus)

            CASE (0)
              IF (MType .NE. 0) THEN
                Call FE_SHELLAlert(Message,Lines,MType)
                IOCODE = 0
              ELSE
                DO 1060 I = 1,Lines
                  WRITE(6,110,IOSTAT=IOCODE,ERR=999) Message(I)
1060            CONTINUE
              END IF

            CASE (1)
              DO 1065 I = 1,Lines
                READ(5,120,IOSTAT=IOCODE,ERR=999) Message(I)
1065          CONTINUE

            CASE (2)
              CINDX = 80
              EndString = .FALSE.
              DO WHILE (EndString .EQ. .FALSE. .AND. CINDX .GT. 0)
                IF (Message(1)(CINDX:CINDX) .EQ. ' ') THEN
                  CINDX = CINDX - 1
                ELSE
                  EndString = .TRUE.
                END IF
              END DO
              WRITE(6,130,IOSTAT=IOCODE,ERR=999) Message(1)(:CINDX)
              READ(5,120,IOSTAT=IOCODE,ERR=999) Message(2)

            CASE (3)
              CINDX = 80
              EndString = .FALSE.
              DO WHILE (EndString .EQ. .FALSE. .AND. CINDX .GT. 0)
                IF (Message(1)(CINDX:CINDX) .EQ. ' ') THEN
                  CINDX = CINDX - 1
                ELSE
                  EndString = .TRUE.
                END IF
              END DO
              WRITE(6,130,IOSTAT=IOCODE,ERR=999) Message(1)(:CINDX)

            CASE (4)
```

```
            Reply = FE_SHELLReply(Message,Lines,MType)
            Message(Lines) = Reply
            IOCODE = 0

          END SELECT

        CASE DEFAULT                              ! Shell mode

          SELECT CASE (MStatus)

            CASE (0)
              DO 1010 I = 1,Lines
                IF (MType .NE. 0) THEN
                  IF (I .EQ. 1) THEN        ! First line, display MType
                    OutWinString = MHeader(MType)//Message(I)
                  ELSE
                    OutWinString = MHeader(0)//Message(I)
                  ENDIF
                ELSE
                  OutWinString = Message(I)
                END IF
                WRITE(6,110,IOSTAT=IOCODE,ERR=999) OutWinString
1010          CONTINUE

            CASE (1)
              DO 1020 I = 1,Lines
                READ(5,120,IOSTAT=IOCODE,ERR=999) Message(I)
1020          CONTINUE

            CASE (2)
              CINDX = 80
              EndString = .FALSE.
              DO WHILE (EndString .EQ. .FALSE. .AND. CINDX .GT. 0)
                IF (Message(1)(CINDX:CINDX) .EQ. ' ') THEN
                  CINDX = CINDX - 1
                ELSE
                  EndString = .TRUE.
                END IF
              END DO

              WRITE(6,130,IOSTAT=IOCODE,ERR=999) Message(1)(:CINDX)
              READ(5,120,IOSTAT=IOCODE,ERR=999) Message(2)

            CASE (3)
              CINDX = 80
              EndString = .FALSE.
              DO WHILE (EndString .EQ. .FALSE. .AND. CINDX .GT. 0)
                IF (Message(1)(CINDX:CINDX) .EQ. ' ') THEN
                  CINDX = CINDX - 1
                ELSE
                  EndString = .TRUE.
```

```
              END IF
            END DO

            WRITE(6,130,IOSTAT=IOCODE,ERR=999) Message(1)(:CINDX)

          CASE (4)
            CINDX = 80
            EndString = .FALSE.
            DO WHILE (EndString .EQ. .FALSE. .AND. CINDX .GT. 0)
              IF (Message(Lines)(CINDX:CINDX) .EQ. ' ') THEN
                CINDX = CINDX - 1
              ELSE
                EndString = .TRUE.
              END IF
            END DO
            Message(Lines) = Message(Lines)(:CINDX)//' (y/n)'
            CINDX = CINDX + 6
            DO 1030 I = 1,Lines-1
              IF (MType .NE. 0) THEN
                IF (I .EQ. 1) THEN        ! First line, display MType
                  OutWinString = MHeader(MType)//Message(I)
                ELSE
                  OutWinString = MHeader(0)//Message(I)
                END IF
              ELSE
                OutWinString = Message(I)
              END IF
              WRITE(6,110,IOSTAT=IOCODE,ERR=999) OutWinString
1030          CONTINUE
            WRITE(6,130,IOSTAT=IOCODE,ERR=999) Message(Lines)(:CINDX)
            READ(5,120,IOSTAT=IOCODE,ERR=999) Message(Lines)

        END SELECT

      END SELECT

999   ProcessIOString = IOCODE
      IF (IOCODE .NE. 0) THEN
        WRITE(6,135) IOCODE
      END IF

110   FORMAT(1x,A)
120   FORMAT(A)
130   FORMAT(1x,A,1x,$)
135   FORMAT(1x,'-PROCESSIOSTRING-IO ERROR:',1x,I6)
! ---------------------------------------------------------------------
!
!     End of ProcessIOString
!
! ---------------------------------------------------------------------
      RETURN
```

```
END
```

4. Command Processing

A. Usage

These functions provide the “Command line” functionality to the Shell.

A.1 PROCESSCMDSTRING

Function Prototype, CHARACTER*(*)

CHARACTER*8	PROMPT
LOGICAL	CBEMPTY

Assign a string to PROMPT

```
CMD = PROCESSCMDSTRING(PROMPT, CBEMPTY)
```

PROCESSCMDSTRING extracts the next command from the command buffer or displays the command prompt and waits for a command to be typed.

Argument Definitions

PROMPT	[in]	A CHARACTER variable representing the “Command line” prompt to display if there are no commands waiting in the buffer.
CBEMPTY	[out]	If the command buffer is empty, this logical is TRUE

A.2 GETMENUCOMMAND

Function Prototype, SUBROUTINE

CMD_STRUC	COMMANDS
CHARACTER*8	PROMPT
INTEGER*4	STATEMENT

Assign values to COMMANDS and PROMPT

```
CALL GETMENUCOMMAND(COMMANDS, PROMPT, STATEMENT)
```

GETMENUCOMMAND uses PROCESSCMDSTRING to either prompt for a command or get a command from the buffer.

Argument Definitions

COMMANDS	[in]	The valid command record for the current Shell mode (see appendix II for a description of the command record.)
PROMPT	[in]	A CHARACTER variable representing a custom prompt if one is to be used.
STATEMENT	[out]	This INTEGER variable represents the command index within the command record for the command that was typed or extracted from the buffer.

B. General Remarks

Under most circumstances, the PROCESSCMDSTRING function will not have to be called directly, but it is available for rare situations when a Shell module may have to directly access the command buffer. (Although, this

should be avoided if possible). This function is used by other Low Level routines like GETMENUCOMMAND, GETREALINPUT, etc.

GETMENUCOMMAND is used in all of the High Level Shell modules that have a valid command record defined (The command record is discussed in detail in appendix II). This subroutine also handles cases where a command is a global command or the command is invalid. If it is desired to just use the standard Shell prompt ("Command>"), then the PROMPT argument should be left empty (ie " ").

C. Functional Description

The PROCESSCMDSTRING function first determines if the command buffer is empty or not and sets the CBEMPTY variable accordingly. If the PROMPT argument is empty (ie. it contains all spaces), then the function attempts to extract the next command from the buffer. If it is successful, then it returns that command. If the command buffer is empty, then it returns "CBEMPTY". When the PROMPT argument contains ASCII characters, the function also attempts to extract a command, but if the buffer is empty in this case, it displays the prompt and waits for user input. If no input is given (the user pressed return), then PROCESSCMDSTRING returns "<CR>".

The GETMENUCOMMAND procedure first determines if a custom prompt is to be used or if the standard prompt is desired. It then invokes the PROCESSCMDSTRING function using the requested prompt to get a command. Once a command is returned, it checks it with all of the valid commands in its COMMANDS record. If a match is found, then it returns the command index in the STATEMENT argument. If a match is not found, then it attempts to find a match with the global command record, and if one is found, calls SHELL_GLOBALS to execute the command. If no match is found with the global commands, then it indicates an invalid command error and prompts for another command. This procedure also determines if a menu needs to be written to the screen based on whether or not the buffer is empty and a command that has a submenu was executed.

D. Errors and Restrictions

It is very tempting to send a literal expression in place of the PROMPT argument when calling GETMENUCOMMAND and PROCESSCMDSTRING. At this time, there is a bug which causes unusual characters to be printed when this is done. Therefore, this should be avoided. Note also that GETMENUCOMMAND only stops executing when a valid command is found, a global command is found, or the user hits the return key at a command prompt. In the case where a valid command is typed, GETMENUCOMMAND returns the command index in the STATEMENT argument. When a carriage return is detected, it takes the number of valid commands in the COMMANDS record, adds 1, and returns this in the STATEMENT argument. If a global command is executed, then it takes the total number of valid commands in the COMMANDS record, adds 2, and returns this value in the STATEMENT argument. The calling Shell modules should contain logic to deal with these two special cases as well.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ----------------------------------------------------------------------------
!
!        Module Name: ProcessCmdString
!        Called by: GetMenuCommand,other user input routines
!        Calls to: none
!
! ----------------------------------------------------------------------------
      FUNCTION ProcessCmdString(Prompt,CBEmpty)
```

```
! ---------------------------------------------------------------------
!
!       Function:   In charge of writting the command prompt to the
!                 screen (whether we are in Shell mode or FSGI mode)
!                 and returning the user's input, or just filters the
!                 current command from the Command Buffer, (If it is
!                 not empty)
!
!       CallerID: 0 = Shell mode, 1 = FSGI mode
!
! ---------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                          By
!  APR 02 1996        Created                                John Bolling
!  APR 14 1996        Added logic to support the NULL prompt argument,
!                  in which case the command buffer is checked only. J.B.
!  MAY 16 1996        Added Did_Menu LOGICAL to SHELLPARMS Common
!                  for better communication between the Shell interface
!                  and the GetMenuText module.                      J.B.
!
! ---------------------------------------------------------------------
      IMPLICIT NONE
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
      CHARACTER*(*) ProcessCmdString
      CHARACTER*8 Prompt
      LOGICAL CBEmpty
      INTEGER Cindex
! ----------------------------Shell Parameters--------------------------
      INTEGER CallerID,Mode
      LOGICAL Do_Menu,Did_Menu
      CHARACTER*80 CommandBuffer
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / SHELLPARMS / CallerID, Mode, Do_Menu, Did_Menu,
     .                      CommandBuffer
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
! See if the Command buffer is empty or not
      IF (CommandBuffer(1:1) .EQ. ' ') THEN
        CBEmpty = .TRUE.
```

```
      ELSE
        CBEmpty = .FALSE.
      END IF

          IF (CBEmpty .AND. (Prompt .NE. ' ')) THEN
            Cindex = INDEX(Prompt,' ') - 1
            IF (Cindex .GT. 0) THEN
              WRITE(6,110) Prompt(:Cindex)
            ELSE
              WRITE(6,110) Prompt
            END IF
            READ(5,120) CommandBuffer
          ELSE
            IF (CBEmpty .AND. (Prompt .EQ. ' ')) THEN
              ProcessCmdString = 'CBEMPTY'
              GOTO 999
            END IF
          END IF

          Cindex = INDEX(CommandBuffer,' ')
          IF (Cindex .EQ. 1) THEN
            ProcessCmdString = '<CR>'
          ELSE
            ProcessCmdString = CommandBuffer(1:(CIndex-1))
          END IF

! left justify the rest of the text in command buffer.

      CommandBuffer = CommandBuffer(Cindex:)
      CommandBuffer = ADJUSTL(CommandBuffer)

! See if the Command buffer is empty or not
      IF (CommandBuffer(1:1) .EQ. ' ') THEN
        CBEmpty = .TRUE.
        IF (.not. Did_Menu) Do_Menu = .TRUE.
      ELSE
        CBEmpty = .FALSE.
      END IF

110   FORMAT(1X,A,$)
120   FORMAT(A)
! ---------------------------------------------------------------------
!
!     End of ProcessCmdString
!
! ---------------------------------------------------------------------
999   RETURN
      END
C23456789012345678901234567890123456789012345678901234567890123456789012
! ---------------------------------------------------------------------
!
```

```
!       Module Name: GetMenuCommand
!           Called By: Shell modules
!           Calls to: ProcessCmdString
!
! ---------------------------------------------------------------------
      SUBROUTINE GetMenuCommand(GMC,Custom_Prompt,Statement)
! ---------------------------------------------------------------------
!
!       Function:      For a given simulation mode, this module pro-
!                      cesses user input and matches it with the appro-
!                      priate command
!
!
! ---------------------------------------------------------------------
!
!       Modifications:
!     Date                       Purpose                          By
!  MAR 29 1996        Created based on the Shell version 1.3
!                 GetMenuCmd module. This version is more
!                 compact and allows for multiple commands
!                 to be entered at the prompt.               J.Bolling
!  MAY 16 1996        Added Did_Menu LOGICAL to SHELLPARMS Common
!                 for better communication between the Shell interface
!                 and the GetMenuText module.                      J.B.
!  JAN 20 1997        Replaced the STRING data type Message() with a
!                 CHARACTER*80 type                                J.B.
!
! ---------------------------------------------------------------------
!
!     Glossary Section
!
! ---------------------------------------------------------------------
!                             Global Variables
!
!     Name     |   Type    |           Description
*Do_Diags       LOGICAL     Diagnostics flag
!
!                             Local Variables
!
!     Name     |   Type    |           Description
*Statement      INTEGER     Statement label corresponding to command
*CommandBuffer  CHARACTER*80Holds user Commands until needed.
*Prompt         CHARACTER*8 various display prompts
*Command        CHARACTER*4 Command returned by ProcessCmdString
*ProcessCmdSt...CHAR*4FUN   Command filtering function
*CBEmpty        LOGICAL     Empty status of the command buffer
*Message        CHARACTER   Generic message
*IOStat         INTEGER     Generic Input/Output code (1=input,0=output)
*ProcessIOStr…  INTFUNCTION Generic Input/Output processing function
! ---------------------------------------------------------------------
      IMPLICIT NONE
```

```
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
      CHARACTER*80 SIMPARC80( 10)
      LOGICAL      SIMPARL  ( 30)

      INCLUDE 'Cmd_Structure.txt/LIST'
      RECORD / CMD_STRUC / GMC
      RECORD / CMD_STRUC / GLOBCMD
      CHARACTER*8 Custom_Prompt,Prompt,Prompt_Std,Prompt_Diags
      CHARACTER*4 Command,ProcessCmdString
      PARAMETER (Prompt_Std = 'Command>')
      PARAMETER (Prompt_Diags = 'CmdDiag>')
      LOGICAL Do_Diags,CBEmpty,CmdFound,Use_Custom_Prompt
      INTEGER Statement,IOStat,I
      CHARACTER*80 Message(1)
      INTEGER ProcessIOString
! --------------------------Shell Parameters-------------------------
      INTEGER CallerID,Mode
      LOGICAL Do_Menu,Did_Menu
      CHARACTER*80 CommandBuffer
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / SHELLPARMS / CallerID, Mode, Do_Menu, Did_Menu,
     .                      CommandBuffer

      COMMON / SIMPARS / SIMPARL,SIMPARC80
      COMMON / GLOBALS / GLOBCMD

! ---------------------------------------------------------------------
!
!     Equivalence Section
!
! ---------------------------------------------------------------------

      EQUIVALENCE (SIMPARL(1)    , Do_Diags       )
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------

100   IF (Custom_Prompt .NE. ' ') THEN
        Use_Custom_Prompt = .TRUE.
        Prompt = Custom_Prompt
      ELSE
```

```
        Use_Custom_Prompt = .FALSE.
      END IF

      CmdFound = .FALSE.

      IF (.not. Use_Custom_Prompt) THEN
        IF (Do_Diags) THEN
          Prompt = Prompt_Diags
        ELSE
          Prompt = Prompt_Std
        END IF
      END IF

      Command = ProcessCmdString(Prompt,CBEmpty)

        DO 1010 I=1,GMC.Num_Commands

       IF (Command .eq. GMC.CmdU(I) .or. Command .eq. GMC.CmdL(I)) THEN
             CmdFound = .TRUE.
         Statement = I
             GO TO 20
       END IF

      Statement = I
1010  CONTINUE

      IF (Command .eq. '<CR>') THEN
        CmdFound = .TRUE.
        Statement = Statement + 1
        GO TO 20
      END IF

! Still here huh? check to see if it's a global command

      DO 1020 I = 1,GLOBCMD.Num_Commands

      IF(Command.eq.GLOBCMD.CmdU(I).or.Command.eq.GLOBCMD.CmdL(I)) THEN
            CmdFound = .TRUE.
        Statement = I
        CALL SHELL_GLOBALS(Statement)
        Statement = GMC.Num_Commands + 2
            GO TO 20
      END IF

1020  CONTINUE

! If we're still here, it must have been a bad command

      IF (.not. CmdFound) THEN
        Message(1) =  'INVALID COMMAND ('//Command//')'
        IOStat = ProcessIOString(Message,1,0,4)
```

```
        CommandBuffer = ' '
        GO TO 100
      END IF

! Check the status of the command buffer and see if we need to show
! a menu or not.

20    IF (CBEmpty) THEN
        IF (Command .EQ. '<CR>' .OR. GMC.HasSubMenu(Statement)) THEN
          Do_Menu = .TRUE.   ! going to new menu; show it
          Did_Menu = .FALSE.
        END IF
        IF (Do_Menu .and. Did_Menu) THEN
          Do_Menu = .FALSE.  ! still in same menu; don't show it
        END IF
      ELSE                   ! more commands to do, don't show menu
        Do_Menu = .FALSE.
        IF (GMC.HasSubMenu(Statement)) THEN
          Did_Menu = .FALSE.
        END IF
      END IF

! ----------------------------------------------------------------------
!
!     End of GetMenuCommand
!
! ----------------------------------------------------------------------
999   RETURN
      END
```

APPENDIX IV.

5. Handling Shell Menus

A. Usage

Shell menus are lists that consist of the valid commands for a particular mode, followed by theit descriptions. This section describes how they can be displayed in the shell.

A.1 GETMENUTEXT

Function Prototype, SUBROUTINE
CMD_STRUC COMMANDS

Assign appropriate data to COMMANDS

CALL GETMENUTEXT(COMMANDS)

GETMENUTEXT displays a list of the valid commands followed by their descriptions.

Argument Definitions

COMMANDS	[in]	The valid command record for the current Shell mode. (see appendix II for a description of the command record.)

B. General Remarks

GETMENUTEXT should be called in each Shell module just before GETMENUCOMMAND is called so that the user will be able to see the current commands and their functions. This subroutine will display the menu as needed. As an example, if the command buffer is not empty, then there is no reason for displaying the menu since the user will not be able to interact with it anyway, and the menu is not written to the screen. There are other instances when this subroutine does not show the menu also. These are discussed in more detail in the next section.

There are two logical variables in the SHELLPARMS Common block (See App. I) that GETMENUTEXT depends heavily on. These are the DO_MENU and DID_MENU logicals. The DO_MENU logical tells GETMENUTEXT when a menu needs to be written to the screen. It is handled by the GETMENUCOMMAND subroutine. Once GETMENUTEXT finishes displaying a menu, it sets the DID_MENU logical to TRUE. While the GETMENUCOMMAND procedure is executing, it keeps track of whether or not the command buffer has a command waiting or not. If it does, then GETMENUCOMMAND sets the DO_MENU logical to FALSE. When the buffer is empty, it will set DO_MENU to true if the carriage return was detected or if the HASSUBMENU field in the COMMANDS record for the chosen command is TRUE (indicating that the Shell is going into a new mode), and sets the DID_MENU logical to FALSE. Thus, the next time GETMENUTEXT is called, it will display the menu. If the previous conditions apply (resulting in DO_MENU being TRUE) but the current menu has already been displayed (DID_MENU is TRUE), then GETMENUCOMMAND sets DO_MENU to FALSE so that the same menu will not be displayed again.

C. Functional Description

GETMENUTEXT first checks the status of DO_MENU to determine if any menus should be displayed. If DO_MENU is TRUE. Then it proceeds to format and display the menu based on the COMMANDS record. Any command Headers (See App II for a description of these) for a particular command are set double-spaced and then center justified holding spaces as insignificant (ISPACE = 1). GETMENUTEXT then concatenates each command with its command description (Commands and descriptions are separated by 6 spaces). The resulting string is then

center justified using the GETCENTERJUSTIFY function holding trailing spaces as significant (ISPACE = 0). These steps are repeated for each valid command in the COMMANDS record. If the current Shell mode is the MAIN mode, then GETMENUTEXT also adds the global commands and their descriptions to the menu. Otherwise it adds the "<CR>" and its description as the last line in the menu.

D. Errors and Restrictions

The GETMENUTEXT procedure assumes a screen width (SCR_W) of 80 characters and a menu line width (MENU_W) of 50 characters when centering the command/description lines. Although the menu line width should not be changed, the screen width parameter can be adjusted to achieve better looking menus for different sized monitors or resolutions.

There is also a restriction on the total number of menu lines (including headers and blank lines) that can be displayed by GETMENUTEXT. The current limit is 100.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: GetMenuText
!           Called By: Shell modules
!           Calls to: <Type>Justify routines, ProcessIOString
!
! ------------------------------------------------------------------------
      SUBROUTINE GetMenutext (GMT)
! ------------------------------------------------------------------------
!
!       Function:   Writes the available commands and their de-
!                   scriptions based on the current simulation
!                   mode.
!
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                          By
!  APR 09 1996        Created as a part of the Shell V1.5
!                 interface.                                      J.B.
!  MAY 16 1996        Added Did_Menu LOGICAL to SHELLPARMS Common
!                 for better communication between the Shell interface
!                 and the GetMenuText module.                     J.B.
!  JAN 20 1997        removed the STRING variable Message() and re-
!                 placed with the CHARACTER*80 MessageC()         J.B.
!
! ------------------------------------------------------------------------
!
!     Glossary Section
!
! ------------------------------------------------------------------------
!                           Global Variables
```

```
!
!     Name     |   Type    |           Description
*VARNAME        VARTYPE     VARDESCRIPTION
!
!                             Local Variables
!
!     Name     |   Type    |           Description
*Mode           INTEGER     Operation mode of the simulation
*CallerID       INTEGER     ID of the active interface (0=Shell,1=FSGI)
*Do_Menu        LOGICAL     Display menu status
*Did_Menu       LOGICAL     Menu display successful status
*MessageC       CHARACTER   Generic message
*IOStat         INTEGER     Generic Input/Output code
*GetCENTERJust… CHARFUN     Center-justifies a character string
*ProcessIOStr…  INTFUNCTION Generic text string processor.
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------

! ----------------------------Locals-------------------------------------
      INCLUDE 'Cmd_Structure.txt/LIST'
      RECORD / CMD_STRUC / GMT
      RECORD / CMD_STRUC / GLOBCMD
      INTEGER I,IOStat,MN
      CHARACTER*50 MenuLine
      CHARACTER*80 MessageC(100)
      CHARACTER*6 Cmd_Des_Limiter
      PARAMETER (Cmd_Des_Limiter = '      ')
      INTEGER Menu_W,Scr_W
      PARAMETER (Menu_W = 50)
      PARAMETER (Scr_W = 80)

      CHARACTER*80 GetCENTERJustify
      INTEGER ProcessIOString
! ---------------------------Shell Parameters--------------------------
      INTEGER CallerID,Mode
      LOGICAL Do_Menu,Did_Menu
      CHARACTER*80 CommandBuffer

! ------------------------------------------------------------------------
!
!     Common Section
!
! ------------------------------------------------------------------------
      COMMON / SHELLPARMS / CallerID, Mode, Do_Menu, Did_Menu,
     .                      CommandBuffer
      COMMON / GLOBALS / GLOBCMD
```

```
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      IF (Do_Menu) THEN

      MenuLine = '          '
      MessageC(1) = GetCENTERJustify(MenuLine,Scr_W,10,0)
      MessageC(2) = GetCENTERJustify(GMT.CmdH(0),Scr_W,40,1)
      MenuLine = '          '
      MessageC(3) = GetCENTERJustify(MenuLine,Scr_W,10,0)

      MN = 4        ! starting message index

1049  DO 1050 I = 1,GMT.Num_Commands
        IF (GMT.CmdH(I) .NE. ' ') THEN
          MenuLine = '          '
          MessageC(MN) = GetCENTERJustify(MenuLine,Scr_W,10,0)
          MN = MN + 1
          MessageC(MN) = GetCENTERJustify(GMT.CmdH(I),Scr_W,40,1)
          MN = MN + 1
          MessageC(MN) = GetCENTERJustify(MenuLine,Scr_W,10,0)
          MN = MN + 1
        END IF
        MenuLine = GMT.CmdU(I)//Cmd_Des_Limiter//GMT.CmdD(I)
        MessageC(MN) = GetCENTERJustify(MenuLine,Scr_W,Menu_W,0)
        MN = MN + 1

1050  CONTINUE

      IF (Mode .EQ. 1) THEN          ! Main mode, Show Globals
        MenuLine = '         '
        MessageC(MN) = GetCENTERJustify(MenuLine,Scr_W,10,0)
        MN = MN + 1
        MessageC(MN) = GetCENTERJustify(GLOBCMD.CmdH(0),Scr_W,40,1)
        MN = MN + 1
        MessageC(MN) = GetCENTERJustify(MenuLine,Scr_W,10,0)
        MN = MN + 1
        DO 1060 I = 1,GLOBCMD.Num_Commands
          MenuLine = GLOBCMD.CmdU(I)//Cmd_Des_Limiter//GLOBCMD.CmdD(I)
          MessageC(MN) = GetCENTERJustify(MenuLine,Scr_W,Menu_W,0)
          MN = MN + 1
1060    CONTINUE
        MessageC(MN) = '       '
      ELSE
        MenuLine = '<CR>'//Cmd_Des_Limiter//'Exit From Menu'
        MessageC(MN) = GetCENTERJustify(MenuLine,Scr_W,Menu_W,0)
        MN = MN + 1
        MessageC(MN) = '       '
```

```
      END IF

      IOStat = ProcessIOString(MessageC,MN,0,0)
      Did_Menu = .TRUE.

      END IF
! ---------------------------------------------------------------------
!
!     End of GetMenuText
!
! ---------------------------------------------------------------------
      RETURN
      END
```

6. Processing User Inputs

A. Usage

These functions are generally used to allow data input to the Shell.

A.1 GETREALINPUT

Function Prototype, REAL*4

CHARACTER*(*)	V_NAME
REAL*4	V_VALUE

Store the name of a REAL variable to be changed in V_NAME and its current value in V_VALUE

V_VALUE = GETREALINPUT(V_NAME, V_VALUE)

GETREALINPUT prompts for a new value to set V_NAME to and changes it.

Argument Definitions

V_NAME	[in]	A CHARACTER string containing the name of the REAL variable to change.
V_VALUE	[in]	The current value of the variable represented by V_NAME.

A.2 GETINTINPUT

Function Prototype, INTEGER*4

CHARACTER*(*)	V_NAME
INTEGER*4	V_VALUE

Store the name of an INTEGER variable to be changed in V_NAME and its current value in V_VALUE

V_VALUE = GETINTINPUT(V_NAME, V_VALUE)

GETINTINPUT prompts for a new value to set V_NAME to and changes it.

Argument Definitions

V_NAME	[in]	A CHARACTER string containing the name of the INTEGER variable to change.
V_VALUE	[in]	The current value of the variable represented by V_NAME.

A.3 GETLOGICALINPUT

Function Prototype, LOGICAL

CHARACTER*(*)	V_NAME
LOGICAL	V_VALUE

Store the name of a LOGICAL variable to be changed in V_NAME and its current value in V_VALUE

V_VALUE = GETLOGICALINPUT(V_NAME, V_VALUE)

GETLOGICALINPUT prompts for a new value to set V_NAME to and changes it.

Argument Definitions

V_NAME	[in]	A CHARACTER string containing the name of the LOGICAL variable to change.
V_VALUE	[in]	The current value of the variable represented by V_NAME.

A.4 GETCHARINPUT

Function Prototype, CHARACTER*(*)

CHARACTER*(*) V_NAME, V_VALUE

Store the name of a CHARACTER variable to be changed in V_NAME and its current value in V_VALUE

V_VALUE = GETCHARINPUT(V_NAME, V_VALUE)

GETCHARINPUT prompts for a new value to set V_NAME to and changes it.

Argument Definitions

V_NAME	[in]	A CHARACTER string containing the name of the CHARACTER variable to change.
V_VALUE	[in]	The current value of the variable represented by V_NAME.

B. General Remarks

While these functions could easily be replaced by standard READ and WRITE statements, it is not recommended. These four functions should be used to get their respective user inputs while in the Shell because they have the added advantage of being able to read from the command buffer as well as from the keyboard. They also interpret a carriage return as keeping the current value. In other words, not only do they allow a variable to be changed, but they also provide a useful utility to show what the current value of a variable is. Note also that the nature of these functions allows the user to type more than one command on the same line just like the standard Shell command prompt. It is also possible to send character literals in place of the V_NAME arguments.

C. Functional Description

These functions first try to extract the user input from the command buffer by calling PROCESSCMDSTRING If the buffer is not empty, then they return the value that was extracted. If their is no text waiting in the command buffer, then they format a prompt made up of the variable name that is to be changed followed by its current value contained in parenthesis. For example, if the variable to be changed was an integer (call it J) and its value was 100, then GETINTINPUT would format the prompt as "J (100)>". They then use the PROCESSIOSTRING function to display the created prompt and wait for keyboard input. The string returned from PROCESSIOSTRING is then saved to the command buffer and extracted immediately using PROCESSCMDSTRING. If PROCESSCMDSTRING returns "CBEMPTY" (implying that the user pressed return at the prompt), then the original value is returned. Otherwise the new value (extracted by PROCESSCMDSTRING) is returned.

D. Errors and Restrictions

The current status of these functions is not very "bullet proof" and could cause some errors if the user input is not what the function expects. (That is, if a character string is extracted from the buffer when a real number is expected from GETREALINPUT). It is easy for programmers to avoid this problem, but due to the scriptable nature of the

Shell, it is not guaranteed that one of these cases will not arise when a casual user is typing multiple commands at the prompt. This problem will have to be eliminated in future versions.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: GetREALInput
!           Called By: various REAL input requests
!           Calls to: ProcessIOString,ProcessCmdString
!
! ------------------------------------------------------------------------
      FUNCTION GetREALInput(V_Name,Default_V)
! ------------------------------------------------------------------------
!
!       Function:   Gets a REAL Input from the Shell Window.
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                          By
!  MAR 04 1996        Created                                     J.B.
!  APR 09 1996        Added calls to ProcessIOString and ProcessCmd-
!                     String for compatibility with Shell V1.5    J.B.
!  JUN 06 1996        Any user input requested in this subroutine is
!                     now placed in the Command Buffer first and then
!                     extracted immediately. (This will allow the user
!                     to input multiple numbers or commands at these
!                     prompts also).                              J.B.
!
! ------------------------------------------------------------------------
      IMPLICIT NONE
! ------------------------------------------------------------------------
!
!     Declaration Section
!
! ------------------------------------------------------------------------
      CHARACTER*(*) V_Name
      CHARACTER*32 Prompt_to_User
      CHARACTER*16 Default_V_String,V_Input_String
      REAL GetREALInput,Default_V,V_Input
      INTEGER Cindx,IOStat
      CHARACTER*80 Message(2)
      LOGICAL CBEmpty
      INTEGER ProcessIOString
      CHARACTER*16 ProcessCmdString
      CHARACTER*8 Prompt
      PARAMETER (Prompt = ' ')
! ---------------------------Shell Parameters-------------------------
      INTEGER CallerID,Mode
```

```
      LOGICAL Do_Menu,Did_Menu
      CHARACTER*80 CommandBuffer
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / SHELLPARMS / CallerID, Mode, Do_Menu, Did_Menu,
     .                      CommandBuffer
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      Default_V_String = '                '

      V_Input_String = ProcessCmdString(Prompt,CBEmpty)

      IF (V_Input_String .EQ. 'CBEMPTY') THEN

        Write (Default_V_String(1:12), '(F12.4)') Default_V
        Read (Default_V_String,'(A16)') V_Input_String
        V_Input_String = ADJUSTL(V_Input_String)
        Cindx = INDEX(V_Input_String,' ')
        Prompt_to_User = V_Name//' ('//V_Input_String(:Cindx - 1)//')>'
        cindx = INDEX(Prompt_to_User,'>')

        Message(1) = Prompt_to_User(:Cindx)
        IOStat = ProcessIOString(Message,2,2,0)
        CommandBuffer = Message(2)
        V_Input_String = ProcessCmdString(Prompt,CBEmpty)

      END IF

      Cindx = INDEX(V_Input_String,' ')

      IF (V_Input_String .EQ. 'CBEMPTY') THEN
        GetREALInput = Default_V
      ELSE
        Read (V_Input_String(:Cindx),'(F)') V_Input
        GetREALInput = V_Input
      END IF

! ---------------------------------------------------------------------
!
!     End of GetREALInput
!
! ---------------------------------------------------------------------
      RETURN
      END
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
```

```
! ---------------------------------------------------------------------
!
!       Module Name: GetINTInput
!           Called By: various INTEGER*4 input requests
!           Calls to: ProcessIOString,ProcessCmdString
!
! ---------------------------------------------------------------------
      FUNCTION GetINTInput(V_Name,Default_V)
! ---------------------------------------------------------------------
!
!       Function:   Gets a INTEGER Input from the Shell Window.
!
! ---------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                          By
!  MAR 04 1996        Created                                    J.B.
!  APR 09 1996        Added calls to ProcessIOString and ProcessCmd-
!                     String for compatibility with Shell V1.5   J.B.
!  JUN 06 1996        Any user input requested in this subroutine is
!                     now placed in the Command Buffer first and then
!                     extracted immediately. (This will allow the user
!                     to input multiple numbers or commands at these
!                     prompts also).                             J.B.
!
! ---------------------------------------------------------------------
      IMPLICIT NONE
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
      CHARACTER*(*) V_Name
      CHARACTER*32 Prompt_to_User
      CHARACTER*16 Default_V_String,V_Input_String
      INTEGER GetINTInput,Default_V,V_Input
      INTEGER Cindx,IOStat
      CHARACTER*80 Message(2)
      LOGICAL CBEmpty
      INTEGER ProcessIOString
      CHARACTER*16 ProcessCmdString
      CHARACTER*8 Prompt
      PARAMETER (Prompt = ' ')
! ---------------------------Shell Parameters-------------------------
      INTEGER CallerID,Mode
      LOGICAL Do_Menu,Did_Menu
      CHARACTER*80 CommandBuffer
! ---------------------------------------------------------------------
!
!     Common Section
!
```

```
! ---------------------------------------------------------------------
      COMMON / SHELLPARMS / CallerID, Mode, Do_Menu, Did_Menu,
     .                      CommandBuffer
! ---------------------------------------------------------------------
!
!     Run Section
!
! ---------------------------------------------------------------------
      Default_V_String = '                '

      V_Input_String = ProcessCmdString(Prompt,CBEmpty)

      IF (V_Input_String .EQ. 'CBEMPTY') THEN

        Write (Default_V_String(1:12), '(I12)') Default_V
        Read (Default_V_String,'(A16)') V_Input_String
        V_Input_String = ADJUSTL(V_Input_String)
        Cindx = INDEX(V_Input_String,' ')
        Prompt_to_User = V_Name//' ('//V_Input_String(:Cindx - 1)//')>'
        cindx = INDEX(Prompt_to_User,'>')

        Message(1) = Prompt_to_User(:Cindx)
        IOStat = ProcessIOString(Message,2,2,0)
        CommandBuffer = Message(2)
        V_Input_String = ProcessCmdString(Prompt,CBEmpty)

      END IF

      Cindx = INDEX(V_Input_String,' ')

      IF (V_Input_String .EQ. 'CBEMPTY') THEN
        GetINTInput = Default_V
      ELSE
        Read (V_Input_String(:Cindx),'(I8)') V_Input
        GetINTInput = V_Input
      END IF

! ---------------------------------------------------------------------
!
!     End of GetINTInput
!
! ---------------------------------------------------------------------
      RETURN
      END
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ---------------------------------------------------------------------
!
!       Module Name: GetLOGICALInput
!           Called By: various LOGICAL input requests
!           Calls to: ProcessIOString,ProcessCmdString
!
```

```
! ---------------------------------------------------------------------
      FUNCTION GetLOGICALInput(V_Name,Default_V)
! ---------------------------------------------------------------------
!
!       Function:   Gets a LOGICAL Input from the Shell Window.
!
! ---------------------------------------------------------------------
!
!       Modifications:
!     Date                      Purpose                          By
!  MAR 04 1996        Created                                    J.B.
!  APR 09 1996        Added calls to ProcessIOString and ProcessCmd-
!                     String for compatibility with Shell V1.5   J.B.
!  JUN 06 1996        Any user input requested in this subroutine is
!                     now placed in the Command Buffer first and then
!                     extracted immediately. (This will allow the user
!                     to input multiple numbers or commands at these
!                     prompts also).                             J.B.
!
! ---------------------------------------------------------------------
      IMPLICIT NONE
! ---------------------------------------------------------------------
!
!     Declaration Section
!
! ---------------------------------------------------------------------
      CHARACTER*(*) V_Name
      CHARACTER*32 Prompt_to_User
      CHARACTER*16 Default_V_String,V_Input_String
      LOGICAL GetLOGICALInput,Default_V,V_Input,CBEmpty
      INTEGER Cindx,IOStat
      CHARACTER*80 Message(2)
      INTEGER ProcessIOString
      CHARACTER*16 ProcessCmdString
      CHARACTER*8 Prompt
      PARAMETER (Prompt = ' ')
! ----------------------------Shell Parameters-------------------------
      INTEGER CallerID,Mode
      LOGICAL Do_Menu,Did_Menu
      CHARACTER*80 CommandBuffer
! ---------------------------------------------------------------------
!
!     Common Section
!
! ---------------------------------------------------------------------
      COMMON / SHELLPARMS / CallerID, Mode, Do_Menu, Did_Menu,
     .                      CommandBuffer
! ---------------------------------------------------------------------
!
!     Run Section
!
```

```
! ---------------------------------------------------------------------
      Default_V_String = '                '

      V_Input_String = ProcessCmdString(Prompt,CBEmpty)

      IF (V_Input_String .EQ. 'CBEMPTY') THEN

        Write (Default_V_String(:1), '(L1)') Default_V
        Read (Default_V_String,'(A16)') V_Input_String
        V_Input_String = ADJUSTL(V_Input_String)
        Cindx = INDEX(V_Input_String,' ')
        Prompt_to_User = V_Name//' ('//V_Input_String(:Cindx - 1)//')>'
        cindx = INDEX(Prompt_to_User,'>')

        Message(1) = Prompt_to_User(:Cindx)
        IOStat = ProcessIOString(Message,2,2,0)
        CommandBuffer = Message(2)
        V_Input_String = ProcessCmdString(Prompt,CBEmpty)

      END IF

      Cindx = INDEX(V_Input_String,' ')

      IF (V_Input_String .EQ. 'CBEMPTY') THEN
        GetLOGICALInput = Default_V
      ELSE
        Read (V_Input_String(:Cindx),'(L1)') V_Input
        GetLOGICALInput = V_Input
      END IF

! ---------------------------------------------------------------------
!
!     End of GetLOGICALInput
!
! ---------------------------------------------------------------------
      RETURN
      END
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ---------------------------------------------------------------------
!
!       Module Name: GetCHARInput
!           Called By: various Character string input requests
!           Calls to: ProcessIOString,ProcessCmdString
!
! ---------------------------------------------------------------------
      FUNCTION GetCHARInput(V_Name,Default_V)
! ---------------------------------------------------------------------
!
!       Function:   Gets a Character Input from the Shell Window.
!
! ---------------------------------------------------------------------
```

```
!
!       Modifications:
!     Date                        Purpose                              By
!  MAR 04 1996        Created                                          J.B.
!  APR 09 1996        Added calls to ProcessIOString and ProcessCmd-
!                     String for compatibility with Shell V1.5         J.B.
!  JUN 06 1996        Any user input requested in this subroutine is
!                     now placed in the Command Buffer first and then
!                     extracted immediately. (This will allow the user
!                     to input multiple numbers or commands at these
!                     prompts also).                                   J.B.
!
! ----------------------------------------------------------------------
      IMPLICIT NONE
! ----------------------------------------------------------------------
!
!     Declaration Section
!
! ----------------------------------------------------------------------
      CHARACTER*(*) V_Name
      CHARACTER*32 Prompt_to_User
      CHARACTER*16 Default_V_String,V_Input_String,V_Input
      CHARACTER*(*) GetCHARInput,Default_V
      INTEGER Cindx,IOStat
      CHARACTER*80 Message(2)
      LOGICAL CBEmpty
      INTEGER ProcessIOString
      CHARACTER*16 ProcessCmdString
      CHARACTER*8 Prompt
      PARAMETER (Prompt = ' ')
! ----------------------------Shell Parameters--------------------------
      INTEGER CallerID,Mode
      LOGICAL Do_Menu,Did_Menu
      CHARACTER*80 CommandBuffer
! ----------------------------------------------------------------------
!
!     Common Section
!
! ----------------------------------------------------------------------
      COMMON / SHELLPARMS / CallerID, Mode, Do_Menu, Did_Menu,
     .                      CommandBuffer
! ----------------------------------------------------------------------
!
!     Run Section
!
! ----------------------------------------------------------------------
      Default_V_String = '                '

      V_Input_String = ProcessCmdString(Prompt,CBEmpty)

      IF (V_Input_String .EQ. 'CBEMPTY') THEN
```

```
        Write (Default_V_String(:16), '(A)') Default_V
        Read (Default_V_String,'(A16)') V_Input_String
        V_Input_String = ADJUSTL(V_Input_String)
        Cindx = INDEX(V_Input_String,' ')
        Prompt_to_User = V_Name//' ('//V_Input_String(:Cindx - 1)//')>'
        cindx = INDEX(Prompt_to_User,'>')

        Message(1) = Prompt_to_User(:Cindx)
        IOStat = ProcessIOString(Message,2,2,0)
        CommandBuffer = Message(2)
        V_Input_String = ProcessCmdString(Prompt,CBEmpty)

      END IF

      Cindx = INDEX(V_Input_String,' ')

      IF (V_Input_String .EQ. 'CBEMPTY') THEN
        GetCHARInput = Default_V
      ELSE
        Read (V_Input_String(:Cindx),'(A)') V_Input
        GetCHARInput = V_Input
      END IF

! ---------------------------------------------------------------------
!
!     End of GetCHARInput
!
! ---------------------------------------------------------------------
      RETURN
      END
```

7. Processing File Requests

A. Usage

This section describes how to get the name of a file from the user when data needs to be saved or loaded into memory.

A.1 GETFILENAME

Function Prototype, SUBROUTINE

CHARACTER*20	HEADER, FILENAME
CHARACTER*4	FILE_EXT, FILETYPE

Assign CHARACTER strings to HEADER and FILE_EXT

```
CALL GETFILENAME(HEADER, FILE_EXT, FILENAME, FILETYPE)
```

GETFILENAME either extracts FILENAME from the command buffer or prompts the user to enter a FILENAME.

Argument Definitions

HEADER	[in]	A CHARACTER string describing the type of file that is to be saved or opened.
FILE_EXT	[in]	A CHARACTER string representing the file extension to be appended to the end of FILENAME.
FILENAME	[out]	A CHARACTER string representing the name of the file to be opened or saved.
FILETYPE	[out]	A CHARACTER string representing the type of file to save or open.

B. General Remarks

GETFILENAME provides a "command line" way of saving files by first looking in the command buffer for a filename and any file options. It is also helpful to be able to recognize a file by its name. To accomplish this, GETFILENAME automatically appends the file extension (usually a period followed by a 3 letter abbreviation) represented by FILE_EXT to the end of the filename. The file options supported are "/txt" for ascii text files, "/mat" for matlab workspace files, and "/bin" for binary data files. (Be aware that the module calling this subroutine must execute according to these options). The default option is "/txt". The FILETYPE argument allows the calling subroutine to know what type of file is about to be saved or opened, and is based on the option specified at the command prompt.

Command Line OPTION	FILETYPE
/txt	TEXT
/mat	MATW
/bin	SBIN

The HEADER parameter is optional and can be used to describe the type of file to save or open. It is only used when the subroutine is called and the command buffer is empty. (If the HEADER parameter is not used, send a blank character string in its place).

C. Functional Description

GETFILENAME first attempts to extract the filename from the command buffer. If the buffer is empty, then it formats a prompt containing HEADER concatenated with " Filename (<16 Chars.):" and calls PROCESSIOSTRING to get a filename from the user. It then checks to make sure that the filename entered is less than 16 characters long and re-prompts the user if necessary. If the command buffer is not empty, and GETFILENAME successfully extracted a filename, then it calls PROCESSCMDSTRING again to get any options and sets FILETYPE appropriately. It then appends the appropriate file extension to the filename.

D. Errors and Restrictions

The "/mat" and "/bin" options override the specified FILE_EXT. GETFILENAME uses ".mat" and ".bin" for these filetypes. If GETFILENAME is called when the command buffer is empty, it displays a prompt for a filename. In this case the options cannot be specified and GETFILENAME will assume a filetype of TEXT. If an option is typed at the prompt, GETFILENAME will ignore it.

E. Source Listing

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
! ------------------------------------------------------------------------
!
!       Module Name: GetFilename
!           Called By: Various file I/O requests, Shell mode only
!           Calls to: ProcessIOString,ProcessCmdString
!
! ------------------------------------------------------------------------
      SUBROUTINE GetFilename(Header,File_ext,Filename,Filetype)
! ------------------------------------------------------------------------
!
!       Function:   Reads in a user request for a filename and adds
!                 the appropriate extension.
!
!
! ------------------------------------------------------------------------
!
!       Modifications:
!     Date                        Purpose                           By
!  SEPT 4 1995        Created to allow general FORTRAN compati-
!                     bility for the F18 Simulations.                JB
!  NOV 2  1995        Added Create_MAT_File LOGICAL to determine
!                     which file extension to use. (Matlab data
!                     file extensions of ".MAT" override the de-
!                     fault file exstensions.)                       JB
!  APR 09 1996        Added calls to ProcessIOString and Process-
!                     CmdString for compatibility with Shell V1.5    JB
!  JUN 06 1996        Added additional logic to support the "/mat"
!                     and "/txt" options and removed all references
!                     to the Create_MAT_File logical. This flag will
!                     only be used in the parent subroutines now. Also
!                     added the arguments Filetype and Header.   JB
!
! ------------------------------------------------------------------------
      IMPLICIT NONE
```

```
! ---------------------------------------------------------------------
!
!       Declaration Section
!
! ---------------------------------------------------------------------
      LOGICAL CBEmpty
      CHARACTER*20 Filename, Header
      CHARACTER*16 File,ProcessCmdString
      CHARACTER*8 Prompt
      CHARACTER*4 File_ext,MAT_ext,Filetype
      CHARACTER*2 Space
      INTEGER Indx1, indx2, IOStat,ProcessIOString
      CHARACTER*80 Message(2)
      PARAMETER (Prompt = ' ')
      DATA Space, MAT_ext / '  ','.mat' /
! ---------------------------------------------------------------------
!
!       Run Section
!
! ---------------------------------------------------------------------
      File = ProcessCmdString(Prompt,CBEmpty)

      IF (File = 'CBEMPTY') THEN
        GO TO 10
      ELSE
        GO TO 15
      END IF

10    IF (Header .EQ. ' ') Header = 'Enter'
      Indx1 = INDEX(Header,'  ')
      Message(1) = Header(:Indx1)//'Filename (<16 Chars.):'
      IOStat = ProcessIOString(Message,2,2,0)
      File = Message(2)
      Indx1 = INDEX(File,'/')
      IF (Indx1 .NE. 0) THEN
        File = File(:(indx1-1))
      END IF
15    Indx1 = INDEX(File,Space)
      IF (Indx1 .EQ. 0) THEN
        Message(1) = 'Filename must be less than 15 characters'
        IOStat = ProcessIOString(Message,1,0,4)
        GO TO 10
      END IF
      Indx2 = Indx1 - 1

      IF (.not. CBEmpty) THEN
        Filetype = ProcessCmdString(Prompt,CBEmpty)

        IF (Filetype .EQ. '/txt' .or. Filetype .EQ. '/txt') THEN
          Filetype = 'TEXT'                       ! ascii requested
        ELSE
```

```
          IF (Filetype .EQ. '/mat' .or. Filetype .EQ. '/MAT') THEN
            Filetype = 'MATW'                    ! Matlab requested
          ELSE
            IF (Filetype .EQ. '/bin' .or. Filetype .EQ. '/BIN') THEN
              Filetype = 'SBIN'                  ! Binary requested
            ELSE
              Message(1) = 'Invalid File Type, using ascii'
              IOStat = ProcessIOString(Message,1,0,4)
              Filetype = 'TEXT'                  ! Default file type
            END IF
          END IF
        END IF
      ELSE
        Filetype = 'TEXT'       ! If not specified, use Default file type
      END IF

      IF (Filetype .EQ. 'MATW') THEN
        Filename = File(:Indx2)//MAT_ext
      ELSE
        IF (Filetype .EQ. 'SBIN') THEN
          Filename = File(:Indx2)//'.bin'
        ELSE
          Filename = File(:Indx2)//File_ext
        END IF
      END IF
! ---------------------------------------------------------------------
!
!       END OF GetFileName
!
! ---------------------------------------------------------------------
      RETURN
      END
```

APPENDIX IV.

8. Writing Matlab™ Workspace Files

A. Usage

This subroutine, along with the Matlab External Interface Libraries, allows Shell data to be saved as binary workspace files for platforms running Matlab.

A.1 CREATE_MATFILE

Function Prototype, SUBROUTINE

CHARACTER*20	FILENAME
LOGICAL	FILEEXISTS
CHARACTER*(*)	MATRIXNAME
INTEGER*4	ROWS
INTEGER*4	COLUMNS
REAL*8	MATRIXDATA(*)

Assign values to all arguments, if data is to be added to an existing file or if the file is to be modified, FILEEXISTS should be TRUE, otherwise create a new file with FILEEXISTS = FALSE

```
CALL CREATE_MATFILE(FILENAME, FILEEXISTS, MATRIXNAME, ROWS, COLUMNS, MATRIXDATA)
```

A “FILENAME.mat” file is created containing the matrix MATRIXNAME with data MATRIXDATA.

Argument Definitions

FILENAME	[in]	A CHARACTER variable representing the name of the file to create or modify.
FILEEXISTS	[in]	A LOGICAL variable indicating whether or not the file specified in FILENAME already exists.
MATRIXNAME	[in]	A CHARACTER string representing the name of the matrix to create.
ROWS	[in]	An INTEGER variable representing the number of rows that the matrix has.
COLUMNS	[in]	An INTEGER variable representing the number of columns that the matrix has.
MATRIXDATA	[in]	A DOUBLE PRECISION array containing the matrix data arranged by column.

B. General Remarks

The CREATE_MATFILE subroutine requires the libmat.o or equivalent external interface library for matlab to work. In addition, this library uses some standard C libraries. Depending on the FORTRAN compiler used, these libraries may also have to be linked with the Shell. For more information on Matlab specific details, consult the Matlab External Interface Guide.

C. Functional Description

Depending on the FILEEXISTS parameter, CREATE_MATFILE calls MATOPEN with the filename and either the ‘w’ or ‘u’ parameter to initialize the file. Next a pointer to the ROWS x COLUMNS matrix is created with the mxCreateFull function. The data in MATRIXDATA is then copied to the pointer’s location and the matrix is named MATRIXNAME using the mxCopyReal8ToPtr and mxSetName functions. Finally, the matrix is saved in the file and the file is closed using the MATPUTMATRIX and MATCLOSE functions. Memory cleanup is accomplished

by a call to mxFreeMatrix.

D. Errors and Restrictions

Although the Shell allows as much as one space in a filename, the external interface libraries do not. Therefore, if a space is used in the filename, and CREATE_MATFILE is called, the saved file will be truncated before the space.

E. Source Listing

```
C-----------------------------------------------------------------------
C
C       Module Name: Create_MATFile
C           Called By: LINRIZE, SHELLTESTDB
C           Calls to: Matlab MEX Libraries
C
C-----------------------------------------------------------------------
      SUBROUTINE Create_MATFile(Filename,Fileexist,Matname,Matrows,
     .                          Matcolumns,Matdata)
C-----------------------------------------------------------------------
C
C       Function:   Will initialize and create a Matlab ".MAT" file
C               with input data.
C
C
C-----------------------------------------------------------------------
C
C       Modifications:
C     Date                      Purpose                          By
C  NOV 1 1995        Created                                     JB
C
C-----------------------------------------------------------------------
      IMPLICIT NONE
C-----------------------------------------------------------------------
C
C       DECLARATION SECTION
C
C-----------------------------------------------------------------------
C---------------------------Inputs--------------------------------------
      LOGICAL Fileexist
      CHARACTER*20 Filename
      CHARACTER*(*)  Matname
      INTEGER Matrows,Matcolumns,Matelements
      REAL*8 Matdata(*)
C---------------------------Locals--------------------------------------
      INTEGER matOpen, mxCreateFull, matClose, mxGetPr, matPutMatrix
      INTEGER a, fp, stat
C-----------------------------------------------------------------------
C
C       COMMON/DATA SECTION
C
C-----------------------------------------------------------------------
```

```
C-----------------------------------------------------------------------
C
C       INITIALIZATION SECTION
C
C-----------------------------------------------------------------------

C-----------------------------------------------------------------------
C
C       RUN SECTION
C
C-----------------------------------------------------------------------
      Matelements = Matrows*MatColumns
      IF (.NOT. Fileexist) THEN
        fp = matOpen(filename,'w')
      ELSE
        fp = matOpen(filename,'u')
      ENDIF
      a = mxCreateFull(Matrows,Matcolumns,0)
      CALL mxCopyReal8ToPtr(Matdata,mxGetPr(a),Matelements)
      CALL mxSetName(a,Matname)
      stat = matPutMatrix(fp,a)
      stat = matClose(fp)
      CALL mxFreeMatrix(a)
C-----------------------------------------------------------------------
C
C       END OF Create_MATFile
C
C-----------------------------------------------------------------------
      RETURN
      END
```

VITA

John Glenn Bolling, a native of Danville Virginia, was Born February 4, 1973. He attended the Danville Public Schools and quickly became interested in mathematics, science, and music. After graduating from George Washington High School in 1991, John began attending Virginia Polytechnic Institute and State University as an undergraduate in Aerospace Engineering. During the summer of 1994, he began conducting research into Constrained Control Allocation techniques, and after graduating Cum Laude in 1995, decided to further pursue this field of research by acquiring a Master's Degree. Throughout this research, John has contributed to the aerospace sciences by co-authoring two conference papers and 1 journal publication for the American Institute of Aeronautics and Astronautics.

Mr. Bolling is now working as a Senior Engineer for the Aerodynamics and Flight Control Department of McDonnell Douglas Aerospace, where he hopes to continue his research and development of control allocation techniques.